(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 May 2002 (10.05.2002)

PCT

(10) International Publication Number
WO 02/37406 A1

(51) International Patent Classification[7]: G06K 9/54

(21) International Application Number: PCT/US01/45501

(22) International Filing Date: 30 October 2001 (30.10.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/243,848 30 October 2000 (30.10.2000) US

(71) Applicant: TRANSLATION TECHNOLOGIES, INC. [US/US]; 655 N. Riverpoint Blvd., Suite 418, Spokane, WA 99202 (US).

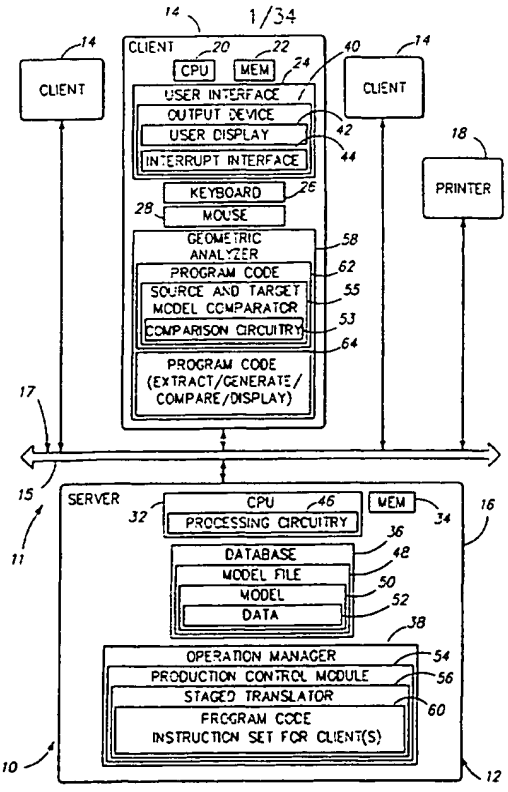
(72) Inventors: JAYARAM, Sankar; 1035 S.W. Meyer Drive, Pullman, WA 99163 (US). JAYARAM, Uma; 1035 S.W. Meyer Drive, Pullman, WA 99163 (US). MCDONALD, Michael, M.; 20415 North Yale Road, Colbert, WA 99005 (US). SORENSON, Robert, L.; 3001 W. Fairway Avenue, Coeur d'Alene, ID 83815 (US). EVANS, Daniel, J.; 3505 Clearlake Court, West Richland, WA 99353 (US). CRAMER, David, M.; 2455 N. Grand Avenue, Pullman, WA 99163 (US).

(74) Agents: GRZELAK, Keith, D. et al.; Wells, St. John, Roberts, Gregory & Matkin P.S., Suite 1300, 601 West First Avenue, Spokane, WA 99201-3828 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AT (utility model), AU (petty patent), AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EC, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

[Continued on next page]

(54) Title: COMPUTATIONAL GEOMETRY SYSTEM, INTERRUPT INTERFACE, GEOMETRIC MODEL COMPARATOR, AND METHOD



(57) Abstract: A computational geometry server (16) is provided that includes a server, a communication link, at least one client (14), and an interrupt interface. The server has processing circuitry and an operation manager. The operation manager is configureS to compare source geometric data in a source geometric model with target geometric data in a target geometric model. Furthermore, the operation manager is operative to identify discrepancies in the geometric data between the source geometric data and the target geometric data. The at least one client communicates with the server over the communication link. The interrupt interface is operative to notify a user of the presence of an inability to automatically generate an accurate representation of the source geometric model in the target geometric model. A method is also provided.

WO 02/37406 A1

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DESCRIPTION

COMPUTATIONAL GEOMETRY SYSTEM, INTERRUPT INTERFACE, GEOMETRIC MODEL COMPARATOR, AND METHOD

Cross Reference to Related Applications

This application claims the benefit of U.S. Provisional Patent Application Serial No. 60/243,848, which was filed on October 30, 2000, and which is incorporated by reference herein.

Technical Field

This invention pertains to drawing conversion. More particularly, this invention relates to computer aided design file translation including a serially progressive interrupt interface having a background operations manager, as well as computer aided design file translation including a geometric model comparator.

Background Art

Practically every product that results from the efforts of generic design or specialized design, such as architectural, electrical and mechanical design, involves the rendering of design drawings. In the last twenty years, nearly all of such drawings have been rendered using a computer aided design (CAD) system with a CAD software program. Typical CAD systems take the form of high-speed workstations or desktop computers that use CAD design software and input devices. These CAD systems generate output in the form of a printed design drawing or an electronic file format that can provide input to a computer aided manufacturing system (CAM).

Since the advent of computerized drawings, numerous computer aided design (CAD) programs have been developed. The translation of data files created by a first computer aided design (CAD) program into one or more data files readable by a second CAD program proves to be a difficult task. However, because of the relatively large number of diverse and competing CAD programs that are available, it is frequently the case that such translations need to be made. A number of underlying reasons exist for performing such translations, including execution of engineering projects that require the services of a main contractor using a first CAD program and a subcontractor using a second CAD program.

A presently existing technique for translating drawings involves the use of a

Document eXchange Format file (DXF file). DXF files comprise an AutoCAD 2-D graphics file format. Numerous CAD systems are provided with the capability to import and export the DXF file format for graphics interchange, in the form "filename.dxf". The DXF file format enables relatively simple files to be saved in DXF format via a first CAD program, and then read via a second CAD program. The DXF file format results in a generally fair translation accuracy where the files are not relatively complex. However, for typically involved applications, DXF file format does not provide a complete solution for translating files.

Accordingly, there exists a present need for an apparatus and method that enables more efficient and more effective translation of CAD files between at least two unique CAD file types, such as from a first CAD file type into a second CAD file type.

Summary

This invention concerns a computational geometry system having an interrupt interface that serially interrupts a user via a user interface at one or more clients associated with a server with a background operation manager and a production control module. The computational geometry system can be provided via a network client/server architecture or via a stand-alone workstation.

This invention also concerns a computational geometry system having a geometric model comparator that identifies discrepancies between target geometric data and source data by comparing the target geometric data with the source geometric data. The computational geometry system can be provided via a network client/server architecture or via a stand-alone workstation.

According to one aspect of this invention, a computational geometry server includes a server, a communication link, at least one client, and an interrupt interface. The server has processing circuitry and an operation manager. The operation manager is configured to compare source geometric data in a source geometric model with target geometric data in a target geometric model. Furthermore, the operation manager is operative to identify discrepancies in the geometric data between the source geometric data and the target geometric data. The at least one client communicates with the server over the communication link. Furthermore, the interrupt interface is provided by one of the at least one client. The interrupt interface is operative to notify a user of the presence of an inability to automatically generate an accurate representation of the source geometric model in the target geometric model.

According to another aspect of this invention, a computational geometry system is provided having a client/server environment, a client, and a server. The client has an interrupt interface. The server communicates with the client via the environment. The server has processing circuitry and an operation manager. The operation manager is configured to compare source geometric data in a source geometric model with target geometric data in a target geometric model. The operation manager is operative to identify discrepancies in the geometric data between the source geometric model and the target geometric model. Furthermore, the interrupt interface is operative to notify a user of the presence of an identified discrepancy in response to comparing the geometric data or encountering a problem in creating the target geometric model.

According to yet another aspect of this invention, an interrupt interface is provided having a server, at least one client, and a production controller. The server has a database that is configured to store input data of a source model. The server also has processing circuitry configured to convert the input data of the source model into corresponding output data of a target model. The server also has a source and target model comparator configured to compare the input data with the corresponding output data, and identify geometric discrepancies between the input and the output data. The at least one client has a user interface, with the client communicating with the server. The production controller has common interface production control software configured to serially interrupt a user via the user interface at one of the clients when at least one geometric discrepancy has been identified.

According to even another aspect, a method is provided for creating a target geometric model from a source geometric model. The method includes providing a server and a client of a computational geometry system having a user interface; extracting source geometric data from the source geometric model; storing the extracted source geometric data in a metafile format; using a target CAD system, generating a target geometric model having target geometric data; detecting at least one of a discrepancy between the models and a problem in generating the target geometric model while generating the target geometric model; and generating an interrupt at the user interface responsive to detecting the at least one of a discrepancy between the models and a problem in generating the target geometric model.

According to one aspect, a method is provided for managing computational geometry system translations. The method includes providing a server and at least one

client within a client/server network environment; receiving source geometric data at the server within a memory; generating target geometric data using the source geometric data; and identifying discrepancies between the target geometric data and the source geometric data by comparing the target geometric data with the source geometric data.

According to another aspect of the invention, a machine-executed method is provided for implementing a geometric conversion on a computer system including an interface. The method includes receiving a source geometric model at a computer; storing the source geometric model in memory of the computer; converting the source geometric model to a target geometric model; extracting source comparison reference data from the source geometric model; extracting target comparison reference data from the target geometric model; and comparing the comparison reference data from one of the source geometric model and the target geometric model with geometry from one of the target geometric model and the source geometric model, respectively, to identify geometric discrepancies therebetween.

According to yet another aspect of this invention, a geometric translation system is provided having processing circuitry, memory, and comparison circuitry. The processing circuitry is configured to generate a target model from a source model. The memory is configured to store the source model and the target model. The comparison circuitry is configured to identify selected points from the source, create corresponding selected points in a target model, and compare the selected points from the source model with the selected points from the target model to identify one or more selected points from the target model that fall outside of a predetermined tolerance range with respect to one or more points from the source model.

According to even another aspect of this invention, a geometric translation system includes memory, processing circuitry, and a geometric model comparator. The memory is configured to receive input data of a source model. The processing circuitry is configured to convert the input data of the source model into corresponding output data of a target model. The geometric model comparator is configured to compare the input data with the corresponding output data and identify geometric discrepancies between the input data and the output data.

According to yet even another aspect of this invention, a machine-executed method is provided for implementing a geometric conversion on a computer system including an interface. The method includes: receiving a source geometric model at a

computer; storing the source geometric model in memory of the computer; converting the source geometric model to a target geometric model; extracting reference data from one of the source geometric model and the target geometric model; importing reference data into one of the target geometric model and the source geometric model; comparing the extracted reference data for discrepancies with geometry from one of the target geometric model and the source geometric model in one of a target CAD system and a source CAD system.

One advantage of the present invention is to provide a novel method and apparatus for computer aided design file translation having an interrupt interface for CAD model (or file) comparison which determines if a translated CAD model (or file) (the output, created by the translation) is geometrically identical to the original source CAD model (or file) from which it was translated.

Another advantage of the present invention is to provide a method and apparatus for computer aided design file translation including a geometric analyzer for CAD file comparison that uses point cloud extraction, including tessellated, surface and edge points.

A still further advantage of the present invention is to provide a method and apparatus for computer aided design model (or file) translation including a geometric analyzer for CAD file comparison that uses a stand-alone point cloud analyzer to increase the speed of the analysis.

One advantage of the present invention is to provide a novel method and apparatus for computer aided design file translation having a geometric analyzer for CAD file comparison which determines if a translated CAD file (the output, created by the translation) is geometrically identical to the original source CAD file from which it was translated.

Another advantage of the present invention is to provide a method and apparatus for computer aided design file translation including a geometric analyzer for CAD file comparison that uses point cloud extraction, including tessellated, surface and edge points.

A still further advantage of the present invention is to provide a method and apparatus for computer aided design file translation including a geometric analyzer for CAD file comparison that uses a stand-alone point cloud analyzer to increase the speed of the analysis.

Brief Description of the Drawings

Fig. 1 is a block diagram overview of a basic system configuration of an exemplary system for implementing computer aided design file translation according to one embodiment of the present invention.

Fig. 2 is a block diagram of a representative geometric data comparison implementation performed via a translation engine of a computational geometry system according to the present invention.

Fig. 3 illustrates data input to a geometric analyzer and respective output in the form of files from the geometric analyzer for a stand-alone workstation embodiment.

Fig. 4 is a perspective view illustrating extraction of point cloud data from a source model which is recreated in a target model wherein distances between respective points and a corresponding surface are measured.

Fig. 5 illustrates an approximation of a cylindrical surface showing respective tessellation triangles when tessellating an exemplary curved, cylindrical surface.

Fig. 6 illustrates an exemplary tessellated curve with tessellation points, or point cloud data.

Fig. 7 illustrates an exemplary geometric model having a matrix of points created along a 3 x 4 surface, showing a 3 x 4 surface point cloud matrix.

Figs. 8A-8B are simplified schematic diagrams illustrating the creation of a hole in a solid cube of material utilizing Pro/E.

Figs. 9A-9D are simplified schematic diagrams illustrating the creation of a hole in a solid cube of material utilizing CATIA.

Fig. 10 is a block diagram illustrating the functional relationship of a staged translator having a translation engine for comparing source geometric data in a source geometric model of a first type with target geometric data of a target geometric model of a second type.

Fig. 11 illustrates mapping of a regeneration process when reordering a Boolean tree.

Fig. 12 illustrates an exemplary Boolean-based CSG tree.

Fig. 13 is a process flow diagram showing the logic processing for performing file translation analysis including interrupt notification via a serial interrupt interface to a user using a user interrupt interface.

Figure 14 is a process flow diagram showing the logic processing for managing computational geometry translations by way of the geometric model comparator.

Fig. 15 is a process flow diagram showing the logic processing for implementing a geometric file conversion on a computer system having an interface.

Fig. 16 is a diagram of an interrupt interface application window within a generic CAD file overlay window of a user display.

Fig. 17 is a diagram of a screen display of a pop-up advanced options application window within a generic CAD overlay window of a user display.

Fig. 18 is a diagram of a screen display in which a pop-up "LogFileViewer" application window is generated within a generic CAD file overlay window in a screen display.

Fig. 19 is a diagram of a screen display for a pop-up graphical user interface feature identification/information application window within an overlay window in the screen display.

Fig. 20 is a simplified schematic diagram illustrating a solid cylinder created by generating B-Rep surfaces in order to enclose a volume.

Fig. 21 is a simplified schematic diagram illustrating a solid model geometry created by extruding a circle section to create a cylinder, sweeping another circle along a trajectory to add a swept feature, and revolving a complex section to create a revolve feature.

Figs. 22A and B are simplified schematic data flow charts illustrating two methods for storing assembly model data.

Fig. 23 is a simplified schematic diagram illustrating a relationship between the location of a hole and a revolve axis of a part identified in a model.

Fig. 24 is a simplified schematic diagram illustrating an exemplary target model having bad points.

Fig. 25 is a simplified schematic diagram illustrating a surface from a source model provided on a target model.

Fig. 26 is a simplified schematic diagram illustrating a source model for a target model having a failed fillet that is annotated with defining parameters and references.

Fig. 27 is a simplified schematic diagram illustrating a source model having a sweep feature in a target model that has failed to build a sweep feature, but has placed construction geometry in the model.

Figures 28 through 76 are flow charts, diagrams, and simplified figures of appended supplemental disclosure materials.

Best Modes for Carrying Out the Invention and Disclosure of Invention

The present invention relates to a computer-implemented technique for notifying a user or operator of a computational geometry system when an identified discrepancy is present within geometric data that is being translated between a source geometric model and a target geometric model. The computational geometry system provides notification to a user when a discrepancy is identified when comparing the geometric data of the source geometric model with geometric data of the target geometric model.

The present invention also relates to a computer-implemented technique for implementing a geometric model comparator. The comparator includes processing circuitry that is configured to generate a target model from a source model. The comparator also includes memory that is configured to store the source model and the target model. Furthermore, the geometric model comparator includes comparison circuitry (provided by processing circuitry and program code) that is configured to identify selected points from the source model, create corresponding selected points in the target model, and compare the selected points from the source model with the selected points from the target model in order to identify one or more selected points from the target model that fall outside of a predetermined tolerance range with respect to one or more points from the source model. In one case, the comparison circuitry implements a forward comparison between the selected points from the source model and the respective selected points from the target model. In another case, this comparison circuitry implements a reverse comparison between the selected points. In even another case, the comparison circuitry implements a bi-directional comparison between the selected points from the source model and the respective selected points from the target model.

A still further advantage of the present invention is to provide a method and apparatus for computer aided design file translation including a geometric analyzer for CAD file comparison that uses a stand-alone point cloud analyzer to increase the speed of the analysis.

Reference will now be made to a preferred embodiment of Applicant's invention. One exemplary implementation is described below and depicted with reference to the drawings comprising a system and method for identifying geometric discrepancies

between input data and output data of a source model and a target model, respectively, such as for a first computer aided design (CAD) model and a second CAD model. While the invention is described by way of a preferred embodiment, it is understood that the description is not intended to limit the invention to this embodiment, but is intended to cover alternatives, equivalents, and modifications such as are included within the scope of the appended claims.

In an effort to prevent obscuring the invention at hand, only details germane to implementing the invention will be described in great detail, with presently understood peripheral details being incorporated by reference, as needed, as being presently understood in the art. Furthermore, U.S. Provisional Patent Application Serial No. 60/243,848, filed October 30, 2000, is incorporated by reference herein.

Figure 1 illustrates a preferred embodiment of Applicant's invention wherein a basic system configuration is provided for comparing input data with output data, identifying geometric discrepancies between the input data and the output data using a geometric model comparator, and interrupting an operator when there is an inability to automatically generate an accurate representation of the source geometric model in the target geometric model, and is identified with reference numeral 10. Geometric model comparator 10 is provided by a computer aided design (CAD) file translation system 12. In one form, system 12 is implemented in a distributed manner across a local area network (LAN) 11, such as a client/server network 13, within a LAN environment. However, it is understood that system 12 can also be implemented on a single, stand-alone workstation, such as on a central personal computer.

As shown in Figure 1, system 12 is implemented across a client/server network 13 having a plurality of clients 14 connected via a communication link 15 with a server 16. In one form, communication link 15 comprises a local area network (LAN) connection 17. According to alternative constructions, a plurality of servers similar to server 16 are included within LAN 11. Further alternatively, server 16 and client 14 can be provided by the same device for the case of a stand-alone workstation. Optionally, for the case of a distributed client/server network implementation server 16 can include a client, similar to client 14. Even furthermore, system 12 can include a single client 14, according to an alternative construction. System 12 also includes a printer 18 provided within LAN 11.

As shown in Figure 1, client 14 in one construction comprises a client computer having a central processing unit (CPU) 20, memory 22, a user interface 24, a keyboard

26, a mouse 28, a geometric analyzer 58, and specialized program code 64 that cooperates with server 16 to extract, generate, compare and display. Although not shown in Figure 1 in order to simplify the drawings, it is understood that all of clients 14 are similarly constructed.

For purposes of this disclosure, the term "client" is understood to include a computer or a workstation, such as a personal computer provided within a client/server environment. A "client" is also intended to include any device present within an environment, such as a LAN environment provided by LAN 11, that includes an interface 24 for enabling a user to interact with client 14 and server 16 such that a user can be notified of important events via an interrupt interface 44. Even furthermore, "client" is intended to encompass the case of a stand-alone workstation, a computer, or a workstation that includes a server similar to server 16. It is further understood that a "client" can include other software components such as a database 36 and an operation manger 38, when constructed as a stand-alone workstation.

Also for purposes of this disclosure, the term "server" is understood to include one or more computers that are located at one or more physical locations within an environment, such as a LAN environment. Furthermore, for purposes of this disclosure, the term "server" is understood to include computers located at one or more physical locations, such as computers distributed about a network or a stand-alone workstation.

More particularly, server 16 includes a central processing unit (CPU) 32, memory 34, database 36, and operation manager 38. Within server 16, CPU 32 includes processing circuitry 46 that communicates with memory 34 in processing data 52 within a database 36 as well as handling program code 60 within operation manager 38. Database 36 includes one or more model files 48 for storing models 50 comprising data 52. Operation manager 38 comprises a production control module 54 including a staged translator 56 in which program code 60 is provided for forwarding one or more instruction sets to clients 14 to cause program code 64 to extract, generate, compare and/or display CAD models (or files).

With respect to client 14, user interface 24 includes an output device 40 which, in one form, comprises a user display 42 and interrupt interface 44. Geometric analyzer 58 includes program code 62, as described below in greater detail.

In order to implement computer aided design (CAD) file translation, geometric model comparator 10 utilizes geometric analyzer 58 to compare CAD files using point

cloud extraction, as discussed below in greater detail. CAD file translation system 12 converts a first CAD file that is associated with a first CAD program into a second CAD file associated with a second CAD program. Geometric analyzer 58 then determines if a translated CAD file is "geometrically identical" to the original source CAD file from which it was translated. For purposes of this disclosure, the term "geometrically identical" is understood to mean geometrically substantially the same, within an acceptable predetermined geometric tolerance. In operation, geometric analyzer 58 implements point cloud analysis, including the identification and analysis of tessellated point cloud types, surface point cloud types, and edge point cloud types, as discussed below in greater detail.

More particularly, geometric analyzer 58 determines if a translated, target CAD model (or file) is geometrically identical to an original, source CAD model (or file). In order to make this determination, geometric analyzer 58 implements a comparison of surface and boundary edges for each respective file. Subsequently, a reverse point cloud analysis is then performed utilizing points that are extracted from the target (or output) file, which is then compared to the source (or input) file.

In summary, the implementation of point cloud extraction entails point cloud analysis which uses points that have been extracted from the source CAD model file. As will be discussed below in greater detail with reference to Figure 4, these points lie either on the surfaces of an individual geometric feature, or they lie on one of the boundary edge curves. These extracted points are then recreated in the target CAD model file, after which distances between the points and the adjacent surfaces and edges are measured. Geometric analyzer 58 then implements point cloud analysis, following implementation of the point cloud extraction, in order to compare the surface and boundary edges of each respective file. It is understood that point cloud analysis of geometric analyzer 58 can be run as a stand-alone apparatus. Alternatively, point cloud analysis can be run as a CAD application that is implemented in accordance with proprietary software and an appropriate software license obtained from a respective CAD application developer.

For the case where a system 12 is implemented as a stand-alone apparatus or workstation, system 12 is not required to use proprietary CAD programs for performing analysis (as sold commercially). Therefore, such a stand-alone apparatus does not require the use of dedicated CAD workstations with proprietary CAD programs that

incur relatively expensive royalties for using such proprietary CAD programs in order to perform analysis. Such a stand-alone CAD workstation has the capability to check points more rapidly because of the ability to optimize such a workstation for a specific purpose. In performing point cloud analysis, the distance between each point and an associated surface or edge is measured. Such measured distances are recognized as being acceptable when they fall within a relatively small (or minimal) threshold value, unless the surface in the output (or target) file has been incorrectly defined, in which case a user at interrupt interface 44 is notified of such a discrepancy.

In operation, geometric analyzer 58 of client 14 is configured to determine if a translated target geometric model file is geometrically identical to the original source geometric model file from which it was translated. In one case, source and target geometric model files each comprise respective CAD files. Geometric analyzer 58 is used to inspect such respective files by measuring the surfaces and boundary edges on all geometric features present in one CAD file, and then compare the respective surfaces and boundary edges to the corresponding surfaces and boundary edges in the other respective file. For example, the measured surfaces and boundary edges from a source geometric model file are compared with the measured surfaces and boundary edges of the respective target geometric model file.

More particularly, in one case geometric analyzer 58 performs a forward inspection by comparing surfaces and boundary edges in a source geometric model file with surfaces and boundary edges in a target geometric model file. Alternatively or additionally, a reverse inspection is performed by geometric analyzer 58 wherein surfaces and boundary edges of a target geometric model file are compared with surfaces and boundary edges of a source geometric model file.

Server 16 of Figure 1 includes database 36 which is configured to store input data of a source model. Processing circuitry 46 is configured to convert the input data of the source model into corresponding output data of the target model. Geometric analyzer 58 includes a source and target model comparator that is configured to compare the input data with the corresponding output data to identify geometric discrepancies between the input data and the output data.

Geometric analyzer 58 enables measurement of such surfaces and boundary edges, then facilitates the reporting of measured and compared results to an operator at a user interface 24 of client 14. Geometric analyzer 58 implements point cloud analysis

in order to perform measurements between the surfaces and boundary edge curves of the source model and the target model. More particularly, geometric analyzer 58 uses points that were previously extracted from the source model file. Such points either lie on the surfaces of the individual geometric features of a model, or such points lie on one of the boundary edge curves of the model. The respective points are then recreated in the target model file, after which distance is measured between each point and respective adjacent surfaces and edges.

In order to increase the probability that the existence of any differences between the source model files and the target model files will be detected, the above process is also implemented in reverse. Accordingly, geometric analyzer 58 extracts points from the target model file, then recreates the points in the source model file. Afterwards, geometric analyzer 58 measures distances between the points and their appropriate surface or boundary edge curve of the target model file and the source model file.

It is understood that point cloud analysis can either be performed using a source CAD system and a target CAD system, or point cloud analysis can be performed using a stand-alone system, such as a workstation, that has been developed specifically as a stand-alone computational geometry system. Such a stand-alone implementation provides an advantage in that special purpose construction can make it relatively faster, thereby enabling a larger number of points to be analyzed within a given amount of time. Such result increases the probability of detecting geometric discrepancies. Additionally, a stand-alone implementation does not require the use of a software license from either a source CAD system provider or a target CAD system provider in order to perform analysis. Accordingly, an improved utilization of resources is provided in that there exists a diminished need to provide for software licensing of source CAD system software or target CAD system software when using a stand-alone conversion system.

Figure 2 illustrates the translation process between a source CAD file and a target CAD file utilizing the computer aided design file translation system of Figure 1. More particularly, translation engine 57 of staged translator 56 (see Fig. 1) receives a source CAD file 63 by way of a secure file transfer 61 from a customer 59. Translation engine 57, by way of a process implemented for the computer aided design file translation system of the present invention, converts source CAD file 63 into a target CAD file 65. The target CAD file 65 is then transferred via a secure file transfer 67

back to customer 59. One technique for implementing the secure file transfer 61 and 67 entails utilizing secure file transfer over the Internet by uploading and downloading files securely over the Internet via one or more secure servers.

Upon uploading a source CAD file 63 to a translation provider's web site, a job description profile is preferably generated. File 63 is then passed to a pre-processing station, configured as a design intent source analyzer 88, which performs design intent analysis, and identifies any pre-processing tasks that are required in order to prepare the file for a translation production line.

Subsequent to processing, resulting file geometry data is then converted into a target CAD file format utilizing native-to-native geometry converter 90. After such conversion, the target CAD model file is compared to the source CAD model file using a mirror-model (or geometric model) comparator 92. Any discrepancies found as a result of source-target file comparison are then conveyed to a human translator in the form of a text report and a visual display that is provided in the target CAD model. The target CAD file is then modified via an operator-assisted conversion process 94 in order to eliminate any discrepancies. The file is then re-compared to the source CAD model as a final quality assurance (QA) 96, and is then sent back to the customer via the Internet.

As shown in Figure 3, all of the surfaces and associated trim curves are required to be extracted from both the source CAD model file and the target CAD model file, after which the extracted associated trim curves are stored in geometric analyzer 58 (of Fig. 1). More particularly, surface data 110 from the source file and surface data 112 from the target file, along with trim curve 114 from the source file and trim curve data 116 from the target file, are input into geometric analyzer 58. Source point cloud data 118 is then analyzed against target surface/trim curve data 110, 114, and target point cloud data 120 is analyzed against source surface/trim data 112, 116, as shown in Figure 3. Bad point cloud files 122, 124 are then imported into respective CAD model files in order to display them to an operator or user, typically via interrupt interface 44 of client 14 (of Fig. 1). Utilizing the CAD software's internal geometry analysis tools, an operator can measure the distance from each individual point to an associated edge or curve in order to ascertain the magnitude and cause of a deviation. Furthermore, an analysis algorithm 128 of geometric analyzer 58 generates a point cloud analysis report 126.

In order to implement point cloud extraction, geometric analyzer 58 (of Figs. 1 and 3) performs three different types of point cloud analysis: tessellated point cloud analysis; surface point cloud analysis; and edge point cloud analysis. A description of each of these different types of point cloud analysis is described below with reference to Figures 4-7.

Operation manager 38 includes a native-to-native geometry conversion system within program code 60. When such system translates a CAD model file, each geometric feature is scrutinized using translator software within staged translator 56; namely, via program code 60. The translator software is operative to extract point cloud data by forwarding an instruction set to clients 14 that cooperates with program code 64 for directing extraction of point cloud data. In order to assure that points are extracted accurately, the distance between points and their corresponding associative entities (surfaces and curves) in the source model file are measured using extraction software within program code 64. All points that do not lie directly on a surface or an edge curve are then deleted. This step is necessary in order to protect against algorithmic errors that are potentially present in CAD system software. Remaining points are then stored in a file which is labeled according to type: tessellated; surface; or edge. In a similar manner, when a file translation has been completed, point cloud data is extracted from the target file and is then stored in memory 22.

System 12 implements point cloud analysis by way of geometric analyzer 58. When a user or operator of client 14 runs geometric analyzer 58, appropriate point cloud files are read by geometric analyzer 58. Such implementation occurs whether system 12 is implemented across a client server network environment or via a stand-alone workstation. The points are then compared against their corresponding entities by measuring the distance between each respective point and the nearest edge or surface that is present in the other CAD file. Points which have been extracted from the source are then measured in the target file, and points extracted from the target are measured in the source file. Surface points are then measured distance-wise against surfaces, and edge points are measured against edge curves.

If it is found that the distance between each individual point and its associated entity is less than a specific predetermined tolerance value that is required for a geometric feature, then the point is deleted from the model file. For example, the distance between a point and an edge curve in a source file is compared with the

distance between the respective point and the respective edge curve in a target file. If the difference between such measured values deviates more than .05 mm (a specific predetermined tolerance value based upon a unit of measure), then the point is not deleted from the model file because it is found to be outside the tolerance range.

When geometric analyzer 58 has measured all of the points in the point cloud file(s), the points that are left over in the model file are points that do not lie directly on (or within a tolerance range) of any surface or edge. Such points represent differences in the geometry between the two files. The coordinates of these points are then saved in a separate file that is appropriately labeled "Bad Point File". Where a forward check is performed, a bad point file is created from the source. Where a reverse check is performed, a bad point file is created from the target. The distance from this point to the associated curve or surface comprises the magnitude of the deviation in geometry at that specific point in space.

Figure 4 illustrates point cloud data that is extracted from a source model 45 and recreated in a target model 45'. The distance between respective points 47 and 47' on the corresponding surface is then measured. The resulting deviation from point 47 to the associated curve or surface (of source model 45 and target model 45') is the magnitude of the deviation geometry at that specific point in space.

Geometric analyzer 58 also identifies different point cloud types. Each of the three previously identified types of point cloud data is extracted separately utilizing a CAD file translation engine provided by staged translator 56; namely, via program code 60 which sends instructions to clients 14. Each type of point cloud data is described below in greater detail. Geometric analyzer 58 uses each point cloud type in order to maximize the advantages of each, while minimizing relative disadvantages. Geometric analyzer 58 uses a tessellated point cloud type in order to quickly find missing features and gross differences in relative geometry. Surface point cloud types are used to very accurately determine differences between surfaces, as well as to verify the dimensional integrity of features. Edge point cloud types are used to verify boundary accuracy and can provide detailed correction information to an operator.

Table 1 - A Comparison of Point cloud Types

Type	Advantages	Disadvantages
Tessellated	• Automatically increases point density for areas of high curvature • Quickly finds missing features • Quickly finds gross geometry errors/differences	• May detect differences on boundary edges • Difficult to create correcting geometry from these points (need edge points to build boundaries) • Can miss subtle geometry errors/differences
Surface	• Very accurate for surface error detection • Quickest analysis to perform • Detects surface differences between the boundary curves	• May not detect differences on boundary edges • Difficult to create correcting geometry from these points (need edge points to build boundaries)
Edge	• Detects boundary curve differences • Facilitates correction geometry creation (especially if used with surface Point cloud)	• Longer analysis time • Does not detect differences on surfaces between the edge curves

Tessellated Point Cloud

Referring to Figure 5, tessellation comprises the process of generating flat triangles 70 that approximate a curved surface of a modeled object. This process typically involves the creation of a series of points 66 that lie on a surface 68. Subsequent to generating points 66, points 66 are joined into a mesh of triangles 70. Accordingly, the vertices 69 of the triangles 70 lie on surface 68, whereas the edges 71 of the triangles may deviate from the surface.

The density of points 66 on cylindrical surface 68, the spatial distribution of points 66, and the number of triangles 70 that are created are often controlled using two parameters: first, chord length; and secondly, angle deviation. These parameters are best described below using the example of a tessellated curve (see Fig. 6).

In Figure 6, a tessellated curve 72 for a boundary curve 73 of a geometric model is shown with a plurality of tessellation points 74-78. As shown, chord length is the maximum distance allowed between a particular line segment 80 and boundary curve 73. The angle deviation (or angle control) defines a maximum allowed angular deviation of the edges of the triangles from a tangent to the curve at the vertex. More particularly, as the curvature of the edge curve becomes greater, the individual line segments that make up a tessellated surface will inherently become smaller. This is necessary in order to maintain the specified chord length and angle deviation. Accordingly, this automatically increases the number of points generated in regions of high curvature. Similar to tessellated curves, tessellated surfaces are still controlled by a chord height, and angular control. Chord height is the largest distance that any given triangular surface is allowed to deviate from a surface being approximated. As a result, the "point cloud" used to create the vertices of all of the triangular surfaces automatically becomes denser in areas of relatively high curvature.

Most presently commercially available CAD software is provided with a tessellation function built therein. For example, Pro/E includes such a tessellation function. In many cases, these tessellation functions can simply be called upon to create points at all of the vertices of a triangular surface in order to generate a tessellated point cloud. However, the process of locating or finding points (triangle vertices) on surfaces often involves mathematical techniques with built-in tolerances. As a result, these tolerances sometimes lead to the creation of tessellated points that fall outside of the desired surface. A tessellated point extractor, configured to extract tessellation point, provided within geometric analyzer 58 analyzes all the points which are created using the tessellation engine of the CAD system, and automatically rejects all points that are not provided exactly on the surface of the model.

Surface Point Cloud

As shown in Figure 7, surface point cloud data is extracted by stepping along each individual surface in a U parameter direction and a V parameter direction, creating points at predetermined parameter intervals. Such a method creates a matrix of points 82 along a surface, such as surfaces 84 and 86. For example, surface points 82 are shown evenly distributed about each of surfaces 84 and 86 of an object in Figure 7.

Edge Point Cloud

Although not shown in a specific figure, by stepping along the boundary edges of each face of a solid (such as the faces of a solid shown in Fig. 7) and creating evenly dispersed points that lie on the edge, edge point cloud data can be generated. The respective faces of a solid, by way of example, can comprise trimmed surfaces that form a bounding surface for the volume of the solid. The edge surfaces are typically parametric curves. The resulting evenly dispersed points are generated by stepping along the curve parameter using a predetermined parameter interval.

Point Cloud Analysis

Once all the point cloud data has been generated with geometric analyzer 58 using the above-described techniques, three-dimensional coordinates for the resulting point cloud data are stored in files according to type. By using presently available programming interface functions of each respective CAD software, these points are created in an appropriate CAD model file. For example, API functions are available for use. The tessellated and surface point clouds are then analyzed by measuring the distance from each individual point to the nearest surface. Edge point clouds are measured to the nearest edge curve. If the resulting distance is equal to zero, or is smaller than a tolerance that has been previously specified in geometric analyzer 58, then the point is deleted from the model. Any points remaining in the model file after analysis is complete are then classified as geometry errors and their coordinates are stored in a separate geometry error file such as bad point files 122 and 124 of Figure 3.

Because the most common type of surface encountered is a surface that is bounded by three or more edges, the number of edge points and the number of edges in a model can quickly become very large. In order to determine which edge provides a nearest edge, geometric analyzer 58 is required to measure the distance from each edge point to each edge within a given model. In models having relatively large numbers of surfaces, edges and points, this analysis can be relatively intensive and time consuming.

Accordingly, in order to speed up edge analysis, the number of edges required to be analyzed against a given point is reduced by measuring only to those edges in which a pre-sized bounding box contains the point being analyzed. More particularly, a bounding box is provided having a dimensional tolerance value in which the bounding box is the smallest three-dimensional box that can contain the entire edge curve.

Accordingly, the above techniques greatly reduce the number of times that the analyzer is required to measure the distance from a given point to a given edge within a model.

Point Cloud Analysis (Stand-Alone Analyzer)

According to an alternative construction where system 12 is implemented within a stand-alone workstation, system modifications are made in order to speed up analysis. In the past, the use of a CAD system's internal tools to measure distances from points to curves and surfaces has typically been a relatively slow process. A slow process results because the analysis tends to be subject to the speed limitations of the CAD software itself. The CAD software is typically busy doing several other operations; for example, one operation entails displaying a complex image to an operator. Accordingly, relatively slow operating speed can result from speed limitations and other operations.

For the case of a stand-alone workstation version of system 12 (and geometric analyzer 58), increases in speed of analysis are implemented by modifying the implementation procedure. Accordingly, point cloud densities can be increased, which in turn improves the accuracy of analysis.

In the case of a stand-alone workstation (and stand-alone geometric analyzer), the same analysis is implemented as was performed for an internal CAD system construction as above described. However, there is a difference in that the surfaces of all the geometric features from the source model and the target model must be extracted for use by the analyzer. The geometric analyzer does not have to display any imagery to the operator, nor does it have to perform any other CAD related functions. Therefore, the geometric analyzer can be optimized for efficient measurement.

The extraction of surfaces from the source CAD system and the target CAD system is done by extracting NURBS surfaces (as discussed below), and their associated trim curves.

Design Intent Source Analysis

With reference to design intent source analyzer 88 (of Fig. 2), there are three purposes for design intent source analysis: First, design intent source analysis is performed to evaluate the "manufacturing design intent" of the source CAD file to assist in the feature mapping between the source and target CAD systems. Secondly,

design intent source analysis is performed to provide process mapping and routing details to the translation production line. Finally, design intent source analysis is performed to predict the cost of the translations and to convey a quotation back to a customer that has requested translation services.

Design intent is conveyed and interpreted by evaluating the methods used to define the geometry in the source file, and determining which methods a designer would use in order to create the same geometry in the target CAD system. A method of creating a specific feature may be perfectly natural to a designer using a source CAD application. However, that method may be very unnatural to a designer using the target CAD system. After performing design intent analysis, such methods are identified, and a natural method mapping can occur.

For example, in Pro/E a designer would create a hole by inputting the hole command, specifying the location of the hole, and the size (or dimensional) parameters of the hole, as shown in Figure 8. Figure 8A illustrates a model of a solid cube comprising solid base material. Figure 8B illustrates a hole which has been inserted on a face of the cube by a Pro/E designer who has created a hole in the solid material of the cube by specifying the surface and location for the hole, and the hole dimensional parameters. In contrast, Figures 9A –D illustrate how a CATIA designer would create a hole by creating a solid cylinder having a desired dimension, locating the cylinder and orienting it relative to the solid cube in which it is desired to put a hole, and then subtracting the solid cylinder from the base solid cube material. Accordingly, the same function in CATIA is accomplished by creating a solid cylinder, locating and orienting the cylinder relative to the cube model, and subtracting the solid cylinder from the remainder of the cube model. Figure 9A illustrates the cube comprising a solid base material. Figure 9B illustrates creation of the solid cylinder. Figure 9C illustrates a cylinder, located and oriented in a desired configuration. Figure 9D shows the cylinder after it has been subtracted from the base, or cube. Table 2, below, illustrates in summary a natural method mapping for a Pro/E CAD file feature as well as a CATIA CAD file feature map.

Table 2 - Natural Method Mapping

Pro/E CAD File Feature	CATIA CAD File Feature Map
Hole	Sol. Cyl [] Locate & Orient [] Subtract Sol. Cyl from Base solid

It is probable that some of the geometric features created by the source CAD application are created in such a manner that the target CAD application cannot create the same geometry using the same method of creating it. One reason this happens is because the specific feature functions that were used in the source system do not exist in the target system. Another reason is due to approximation algorithms. Most CAD systems approximate complex geometry using proprietary algorithms. It is very unlikely that two CAD systems will use the same algorithms to approximate the same geometric features. Therefore, differences in the geometry are bound to occur even though the feature in the target file was created using the same methodology as the feature in the source file. Regardless of the cause of any inaccuracy, the solution is the same: A set of features needs to be found in the target CAD system that duplicates the geometry found in the source file. This requires mapping of feature geometry methods to the target CAD system.

In order to accurately predict the cost of converting a CAD model file, there are four pieces of information necessary. First, the size of the file; secondly, the number of geometric features that need to be converted; thirdly, the type of features; and finally, source and target CAD file applications. The source design intent analyzer performs a series of scans on the source file in order to determine this information. First, It counts the number of features in the source model file and parses them into specific feature type categories. Then, the features are classified into categories of a) geometry that always automatically converts over every time; b) geometry that automatically converts over X% of the time; and c) geometry that never automatically converts over and requires operator assistance to complete it. This classification will be specific to the particular source and target CAD systems. Once the features in the source model file are categorized and counted, then simple algorithms can be used to predict the conversion cost.

Native-to-Native Geometry Conversion

According to the present translation engine implementation, the approach to CAD file conversion is based upon the assumption that a purely automated software solution is not possible. Existing technical barriers are numerous, which prohibits a purely automated software solution. Therefore, a technology is needed that integrates the automatic recreation of geometry in the target CAD system.

There are two major steps to the native-to-native geometry conversion process of converter 90 (see Fig. 2). First, there is a source to Applicant's model format (an intermediate format) conversion. Secondly, there is Applicant's model to target file conversion.

Both of these steps are accomplished by utilizing the CAD system's own application program interfaces (API's). The source CAD system's own function calls are then used to extract a list of the geometric features that make up the source model file. These features and their defining parametric data are stored in a series of Applicant's proprietary data formats. The formats are described in the algorithm section below. Utilizing the equivalent function calls for the target CAD system, the geometric features are recreated by reading the geometric feature data from the Applicant's model formats. This provides a "hub and spoke" type of configuration in which once a CAD system is implemented to write to and read from the Applicant's model formats, it is then possible to convert file to or from any other CAD system that is implemented.

In order to facilitate the process of converting from a source file format to Applicant's formats and from Applicant's formats to a target file format, several CAD system specific utilities were developed. These utilities perform specific tasks including but not limited to: extracting mathematical geometric data from the source model file and storing it in Applicant's proprietary format, mapping certain geometry generation function calls from Applicant's formats to the appropriate target CAD system function calls, and determining specific geometric data that is not necessarily provided by the source CAD system, but will be needed in order to generate the equivalent geometry with the target CAD system.

Geometric Model Comparison

Geometric model (or mirror-model) comparison is the process of comparing the source CAD file with the target CAD file to determine if discrepancies exist. The comparison consists of measuring the distances between the corresponding surfaces and boundary edges of the two files. If the target file is a perfect duplicate of the source file, then all of these measurements will yield zero distances.

The process invokes Applicant's point cloud data (or Pixie Dust) analysis process, which creates a series of points that lie on the surfaces and edge curves of each geometric feature of a model. These points are then brought into the target CAD model file and the distance from the points to the corresponding surfaces and curves are measured. Any points that lie on the appropriate surface or edge curve, or within acceptable tolerances, are deleted from the target model. The points that lie outside of the acceptable tolerances are flagged as errors in the model file.

The reverse of this process, where points are extracted from the target model file and are inserted into the source file can also be invoked. This increases the probability of finding all of the discrepancies between the two files and reduces the probability of an error going undetected.

There are three general types of point cloud points generated, which include tessellated points, surface points, and edge points. Tessellated point cloud points use a tessellation routing to determine a series of vertices that are scattered all about a surface.

An operator can then select which type of point cloud analysis to run. The advantages and disadvantages of these three types were described above in Table 1.

The translation production line concept is a process that treats CAD file conversion as a manufactured product. The process applies manufacturing production line techniques to the process of converting CAD file formats.

As shown in Figure 10, staged translator 56 includes a translation engine 57. Translation engine 57 implements conversion/translation and analysis/inspection when converting from a source CAD model (or file) (or a CAD file that a customer wishes to convert from) to a target CAD model (or file) (or a CAD file that a customer wishes to convert to). For example, various different source CAD files and target CAD files are shown in Figure 10. For example, a Pro/E CAD file 100, an SDRC CAD file 101, and a Unigraphics CAD file 102 each comprise source CAD files; whereas, a CATIA CAD

file 103, a SolidWorks CAD file 104, and an AutoCAD file 105 each comprise target CAD files.

The translator implementation of Figure 10 enables a relatively efficient quotation process when the system of the present invention is used to implement staged translation for a third party desiring translation of drawings from a source CAD file into a target CAD file. Such a process entails: reviewing source CAD model geometric features using feature statistics; implementing design intent analysis and feature parsing; and estimating conversion costs based upon the number of features, and the types of features identified.

Figure 11 illustrates the mapping of a regeneration process comprising the reordering of a Boolean tree. Figure 11 illustrates a feature based Boolean tree. Figure 12 illustrates a respective Boolean based CSG tree. By mapping the regeneration process using the feature based tree of Figure 11 and the Boolean based CSG tree of Figure 12, the manner in which errors can be easily and serially displayed to a technician in a serial manner and in a target CAD file are readily apparent. A technician can complete features that the software cannot properly complete (as identified by an interrupt interface). Furthermore, the technician can additionally or alternatively repair any errors that exist within the geometry as identified by the system.

Overall Implementation

In order to implement the computerized design model (or file) translation system features of the present invention, the following methodology is carried out. In Stage 1, a source model file design intent analysis is performed. Subsequently, a Stage 2 analysis generates a duplicate target model file corresponding with the source model file with the pre-existing, source model file. In Stage 3, a mirror-model comparator analysis is implemented by way of a forward check, a backward check, or a forward and backward check. Finally, a Stage 4 operation entails an operator inspecting the resulting analysis file and correcting any errors that are detected, and then rerunning the analysis of Stage 3 by way of a mirror-model comparator.

According to Stage 1, source model file design intent analysis entails receiving a pre-existing, or source model file, then storing the model file in memory. Subsequently, the pre-existing, source model file is opened using a source CAD system. Next, the pre-existing, source model file architecture is evaluated, and the model file is exploded.

Subsequently, construction history indicating how the model was originally graphically built is then examined. Furthermore, comparison reference data (or point cloud data) is then extracted. Finally, the extracted data is stored in a designated metafile format (an intermediate file format).

In Stage 2, the methodology includes launching a target CAD system. Subsequently, a target model file is recreated, duplicating the same process that was used to create the original pre-existing, source model file. Interrupts are subsequently generated, when necessary, indicating to an operator that help is needed in recreating the target model file. For example, an interrupt can be displayed on a user display screen to a user indicating "I need help". Finally, Stage 2 entails an operator "clearing" any interrupts (or removing and fixing the cause) that are presented to the user or operator during recreation of the target model file.

In Stage 3, the methodology entails a forward check and a backward check. In the forward check, points are created in the target CAD file representing the location of edges, and surfaces of the source model. Distance is then measured between the points and the edges/surfaces. Subsequently, points are deleted that fall within a predetermined tolerance. The forward check is finally completed by saving a model file containing "bad" points, or points that fall outside the predetermined tolerance. The backward check is implemented in the same manner as the forward check, except point cloud data is extracted first from the target model, then compared with point cloud data extracted from the source model. A forward and backward check entails performing both checks, then saving "bad" points from both check into a common file.

Stage 4 of the methodology includes an operator inspecting the analysis file resulting from the previous stages. Next, distances are measured between points and edges/surfaces. Subsequently, points that deviate, or fall outside a predetermined value are identified and a determination is made as to whether corrective action is required on the part of the operator. If corrective action is required, the operator then corrects any errors in the analysis file. Finally, the operator reruns the analysis, starting again with Stage 3 in order to determine whether the corrective action has remedied the problem with respect to the geometric data which is requiring corrective action to fall within the predetermined tolerance.

Figure 13 illustrates by way of example one method for creating a target geometric model file from a source geometric model file wherein an interrupt is

presented to a user at a user interface in a timed manner that occurs serially during creation of the target geometric model file and when a discrepancy is detected during creation of the target geometric model file. In this manner, a serial arrangement of interrupts can be presented to a user as the process of creating a target geometric model file progresses from start to finish. In response to each interrupt, the user is given an opportunity to evaluate the discrepancy and correct such discrepancy, after which the check is re-performed in order to confirm compliance of the target data with the respective source data.

As shown in Figure 13, a logic flow diagram illustrates the steps of implementing a serial interrupt interface process when creating a target geometric model from a source geometric model.

In Step "S1", a server and a client of a computational geometry system are provided having a user interface that is used to provide an interrupt to a user. After performing Step "S1", the process proceeds to Step "S2".

In Step "S2", the system receives a pre-existing, source geometric model at the server. After performing Step "S2", the process proceeds to Step "S3".

In Step "S3", the system stores the source geometric model in memory of the server. After performing Step "S3", the process proceeds to Step "S4".

In Step "S4", the source geometric data is extracted from the source geometric model. After performing Step "S4", the process proceeds to Step "S5".

In Step "S5", the system stores the extracted source geometric data in a metafile format. After performing Step "S5", the process proceeds to Step "S6".

In Step "S6", the system, using a target CAD system, generates a target geometric model having target geometric data. After performing Step "S6", the process proceeds to Step "S7".

In Step "S7", the system detects at least one of a discrepancy between the models and a problem in generating the target geometric model while generating the target geometric model. After performing Step "S7", the process proceeds to Step "S8".

In Step "S8", the system generates an interrupt at the user interface responsive to detecting the at least one of a discrepancy between the models and a problem in generating the target geometric model. After performing Step "S8", the process proceeds to Step "S9".

In Step "S9", the system interrupts generation of the target geometric model in response to generating the interrupt. After Step "S9", the process proceeds to Step "S10".

In Step "S10", a user fixes the problem and/or discrepancy. After Step "S10", the process proceeds to Step "S11".

In Step "S11", a user clears the interrupt using the user interface. After Step "S10", the process proceeds to Step "S12".

In Step "S12", a query is made as to whether the model generation process is complete. If the file generation process is determined to be complete, the process is terminated. If not, the process returns back to Step "S6".

According to another implementation, a method is implemented using a computational geometry system to realize an interrupt interface that notifies a user of the presence of any identified discrepancies in response to comparing geometric data between source geometric data and target geometric data in a source geometric model and a target geometric model, respectively. Such a computational geometry system utilizes a server with processing circuitry and an operation manager that is configured to compare source geometric data and a source geometric model with target geometric data and a target geometric model. The operation manager is further operative to identify discrepancies in the geometric data therebetween. A client of the computational geometry system communicates with the server over a communication link. The interrupt interface is provided by one of at least one client. The interrupt interface is operative to notify a user of the presence of any identified discrepancies in response to comparing the geometric data. In one case, the operation manager comprises a medium having program code embodied therein which, when executed by the processing circuitry, translates a source geometric model file received from a source within a client/server environment to a target geometric model file.

In one case, the program code is associated with the processing circuitry in the interrupt interface. The program code includes a production control module that is operative to implement staged translation of the source geometric model file into the target geometric model file.

In one case, staged translation includes the steps of extracting comparison reference data from the source geometric model file and a source CAD system; generating a target geometric model file and a target CAD system; comparing reference

data from the source geometric model file with corresponding reference data from the target geometric model file; and, upon identifying a discrepancy, displaying the discrepancy to an operator at the client with the interrupt interface.

Figure 14 illustrates by way of example one method for managing computational geometry system translations by way of a geometric model comparator that utilizes point cloud features as well as a forward check, a backward check, and/or forward-backward check. In this manner, selected points can be identified from a source model (or file) and corresponding selected points can be generated in a target model (or file). The selected points are compared from the source model with the selected points from the target model in order to identify one or more selected points from the target model that fall outside of a predetermined tolerance range with respect to one or more points from the source model. A geometric model comparator compares the input data with corresponding output data and identifies any geometric discrepancies between the input data and the output data.

As shown in Figure 14, a logic flow diagram illustrates the steps of implementing such a geometric model comparator.

In Step "S1", a server and at least one client are provided within a client/server network environment. After performing Step "S1", the process proceeds to Step "S2".

In Step "S2", the system receives source geometric data including comparison reference data extracted from a pre-existing source model at the server within memory. After performing Step "S2", the process proceeds to Step "S3".

In Step "S3", the system stores the extracted comparison reference data within a meta file format utilized by Applicant's system. After performing Step "S3", the process proceeds to Step "S4".

In Step "S4", the process proceeds by opening the source model using a source CAD system, such as CATIA. After performing Step "S4", the process proceeds to Step "S5".

In Step "S5", the process proceeds by generating target geometric data including comparison and reference data created in a target model using the source geometric data. After performing Step "S5", the process proceeds to Step "S6".

In Step "S6", the process identifies discrepancies between the target geometric data and the source geometric data by comparing the target geometric data with the source geometric data. After performing Step "S6", the process proceeds to Step "S7".

In Step "S7", the process and system notify a user or operator at the client of the presence of the discrepancy. After performing Step "S7", the process proceeds to Step "S8".

In Step "S8", a query is made as to whether the operation of identifying discrepancies is complete. If the process is not complete, the process returns back to Step "S6". If the process is determined to be complete, the process is terminated.

Figure 15 illustrates by way of another example a machine-executed method for implementing a geometric file conversion on a computer system including an interface. In Step "S1", a source geometric model is received at a computer. After performing Step "S", the process proceeds to Step "S2".

In Step "S2", the source geometric model is stored in memory of the computer. After performing Step "S2", the process proceeds to Step "S3".

In Step "S3", the source geometric model is converted into a target geometric model. After performing Step "S3", the process proceeds to Step "S4".

In Step "S4", the source comparison reference data is extracted from the source geometric model, wherein the source comparison reference data comprises point cloud data. After performing Step "S4", the process proceeds to Step "S5".

In Step "S5", target comparison reference data is extracted from the target geometric model. After performing Step "S5", the process proceeds to Step "S6".

In Step "S6", the source comparison reference data is compared with target comparison reference data in order to identify geometric discrepancies. The comparison is implemented by determining whether point cloud data from the target geometric model lies outside of a terminal surface of point cloud data from the source geometric model using a predetermined geometric tolerance. After performing Step "S6", the process proceeds to Step "S7".

In Step "S7", a user or operator is interrupted at the interface of the computer system when a geometric discrepancy is identified and/or a problem is encountered during converting the source geometric model to the target geometric model. After performing Step "S7", the process proceeds to Step "S8".

In Step "S8", a query is made as to whether or not the discrepancy identification is complete. If the discrepancy identification is not complete, the process returns back to Step "S6". If the discrepancy identification is complete, the process is terminated.

Figures 16-19 illustrate by way of example a serial interrupt interface 44 as seen from a client by a user or operator and comprising a generic CAD file overlay, or window, 130. Interrupt interface 44 is provided by way of a user display 42. User display 42 provides one implementation for a user interrupt 24 triggered by detecting a discrepancy or a problem in generating a target geometric model. Such user interface features are implemented on a client/server computer system and are presented below to illustrate one implementation of Applicant's system and method for providing a serially progressive interrupt interface with a background operation manager when translating a source geometric model to a target geometric model using a computational geometry system. According to one implementation, the computer aided design file translation system comprises a serially progressive interrupt interface which is provided by way of a pop-up interrupt interface application window 132 which is displayed within generic CAD file overlay window 130.

As shown in Figure 16, a graphical user interface on a client such as the display screen is used to display serial interrupt interface 44 via displaying interrupt interface application window 132 within overlay window 130. With respect to Figures 16-19, each screen contains icons, or triggers, that may be clicked on in order to select different items and/or in order to navigate between screens.

Figures 16-19 each form diagrams of exemplary screen displays that the geometric model comparator and computer aided design file translation system provides to a user or operator during various steps when serially interrupting the operator when translating a source geometric model file to a target geometric model file and their existent inability to automatically accurately create geometry from the source geometric model file to the target geometric model file. For purposes of this disclosure, it is understood that a screen displays a portion of the computer aided design file translation system, and that the screen includes a window which is a predefined part of virtual space. Accordingly, a screen can include selection buttons, pop-up menus, pull-down menus, icons, links, buttons, scrolling lists, data entry fields, embedded content objects, radio buttons, check boxes, and other usable and selectable items that are capable of being configured or selected with a cursor using a tactile input such as a pointer, a mouse, and/or a keyboard or a button.

Figure 16 illustrates one exemplary interrupt interface application window 132 that pops up within generic CAD file overlay window 130 in response to the

computational geometry system encountering a problem or inability to automatically create an accurate geometry when translating from a source geometric model to a target geometric model. One exemplary problem that creates an inability to automatically create an accurate geometry results when the computational geometry system determines the presence of any identified discrepancies in response to comparing the geometric data between the source geometric model and the target geometric model.

The interrupt interface application window 132 of Figure 16 includes a "Continue Options" field 134, an advanced options field 136, and a "Log File Viewer" field 138. "Continue Options" field 134 includes an import button 140 that selectively configures with a "Next Feature" check box 142 or a "Feature I.D." check box 144. "Feature I.D." check box 144 also includes a data entry field 146 for entering a desired feature identification. Such feature is entered into data entry field 146 after which check box 144 is selected such that the desired feature identification is imported when selecting import button 140. Such action continues operation or stops the translation process back up, starting with the next feature (if check box 142 is selected), or starts the process back up at the identified feature selected by way of check box 144 and data entry field 146. Advanced options field 136 includes an options button 148 that generates pop-up advanced options application window 150 of Figure 17. "LogFileViewer" field 138 includes a start button 152 that launches a pop-up "LogFileViewer" application window 154, as shown in Figure 18.

Figure 17 illustrates the generation of pop-up advanced options application window 150 within generic CAD overlay window 130 of user display 42. More particularly, pop-up application advanced options application window 150 comprises an import single feature field 156, an import feature surfaces field 158, an import single surface field 160, an import section field 162, an import feature edges field 164, an import surface edges field 166, and a restart field 168. Furthermore, window 150 includes a cancel button 170 which triggers closure of window 150.

Import single feature field 156 includes a data entry (or text) field 172 for inputting an import single feature identification number (or alpha-numeric) identifier. Import single feature field 156 also includes a union check box 174, a subtraction check box 176, and an intersect check box 178. Selection of boxes 174-178 in combination with input of an identification number within field 172 enables the selection of that import single feature by way of a union, a subtraction, or an intersection, respectively. After

designation of the appropriate check box and input of the identification number, an import button 180 is selected by a user to import such single feature in the manner designated by the selection of the respective check box amongst check boxes 174-178.

Import feature surfaces field 158 includes a data entry (or text) identification field 182, a volume check box 184, a trim check box 186, and an import button 188. Insertion of data within field 182, selection of the appropriate check box 184 or 186, and selection of import button 188 causes importation of the identified feature surfaces by way of volume or trim, respectively.

Import single surface field 160 includes an import single surface identification data entry (or text) field 190, a trim check box 192, and an import button 194. A user provides an identification data entry number within field 190, selects trim check box 192, and import button 194 in order to cause import of a single surface having the desired identification and respective trim characteristic.

Import section field 162 includes an import section identification data entry (or text) field 196 and an import button 198. A user provides the identification within field 196 and then selects import button 198 in order to import the identified section.

Import feature edges field 164 includes an import feature edges identification data entry (or text) field 200 and an import button 202. A user imports an appropriate identification within field 200 and selects import button 202 in order to import the identified feature edge.

Import surface edges field 166 includes an import surface edges identification data entry (or text) field 204 and an import button 206. A user merely inputs an identification number within field 204 and selects import button 206 in order to import the identified surface edge. Restart field 168 includes a restart job button 208. A user selects restart job button 208 in order to restart the translation process implemented by the computer aided design filed translation system of the present invention as described above with reference to Figures 1-15. Finally, a user merely selects cancel button 170 in order to cancel the advanced options features provided by way of application window 150, thereby closing window 150.

Figure 18 illustrates a screen display in which pop-up "LogFileViewer" application window 154 is generated within generic CAD file overlay window 130 in response to selection of start button 152 of Figure 16. "LogFileViewer" application window 154 enables an operator to view a log file for a specific job that is identified by

way of a job identification (JOBID). Specific feature attributes can be entered via a sequence number data entry field 210, a type data entry field 211, a source identification entry field 212, and a target layer data entry field 213. A creation status data display box 214 generates status information on the creation of the feature attribute. A navigation filter match feature status pull-down menu 216 is also provided. Furthermore, a navigation filter match feature type pull-down menu 218 is also provided. An apply filters button 220 and a toggle status button 222 are furthermore provided. Feature attribute information can be provided by way of a message box 224. Furthermore, an internal display 226 provides feature attribute information therein.

Furthermore, application window 154 includes navigation direction buttons comprising a first button 228, a next down button 229, a "down>>" button 230, a next up button 231, an "up<<" button 232, and a last button 233. Selection of first button 228 grabs the first feature attribute. Selection of button 229 moves to the next down feature attribute. Selection of button 230 moves down to the very bottom of the feature attributes. Selection of next up button 231 moves up one feature attribute. Selection of button 232 moves to the top of the feature attributes list. Selection of last button 233 moves to the very last feature attribute.

A find data entry field 234 enables a user to input a desired feature attribute, after which selection of "Find this I.D." 235 causes the navigation to the feature attribute identification provider within data entry field 234. Likewise, input of data (or an identification number) into data entry field 236 and selection of "Find this Sequence Number" 237 finds the respective identified sequence number. A "Total Number of Features" data field 238 displays the total number of features and a number of features created successfully data field 239 displays the number of features that have been successfully created. Such fields 238 and 239 are provided in a statistics format display 242 within application window 154. An "I Done" button 240 enables the designation that LogFileViewer utilization is done, thereby closing window 154. Selection of a reload button 241 reloads the LogFileViewer, relaunching application window 154.

Figure 19 illustrates a pop-up graphical user interface feature identification/information application window 244 which is displayed within overlay window 130 in response to selection of import button 140 of Figure 16. Application window 244 includes a source feature identification field 246, a feature information field 248, and a translation options field 250. Source feature identification field 246 includes

a source feature identification data entry field 252, a feature type data entry field 254, a Boolean type data entry field 256, and a creation status data entry field 258. Furthermore, field 246 includes a reasons for interrupt scrollable display 260. Feature information field 248 includes a plurality of tab screens such as a summary tab screen 262 comprising a summary of the feature information. A features parameters tab screen 264 includes information on feature parameters. Translation options field 250 includes a continue button 266 that enables continuation of further translation options.

Translation Process Overview –

Translation Process Principles

Applicant has developed a unique approach to the task of translating Computer Assisted Design (CAD) models from one format to another. The process is designed to a) maximize the quality of the translation, b) minimize cost of translation. In order to achieve these two goals it is necessary to ascertain the geometric accuracy of the translation, and perform the translation with a minimization of human interaction time.

As to determining the geometric accuracy of the translation, a method has been developed for comparing a source CAD model (model to be translated), with a target CAD model (the translated model). The objective of this comparison is to locate and convey any differences in geometry that may exist between the two models, and convey them to an operator so that they can be resolved. For purposes of this comparison, the source model geometry is defined to be correct. The comparison method involves comparing all boundary surfaces, their trim curves, and all model edges to one another. This method is useful for locating differences in geometry, missing geometry, or extraneous geometry in the target model.

Human interaction is minimized by providing a systematic process in which operations that can be automated are done so without human input, and the operations that cannot be automated are presented to the human operator in a manner that displays the specific problem concisely and efficiently, and provides the operator with all of the information necessary to quickly arrive at the solution. Operations which can be automated are performed in the "background" while any human intervention operations are performed in the "foreground".

The translation process employs the necessary controllers and systems to notify the human operator in the event it needs assistance to resolve a specific translation

problem, or has incorrectly created the geometry in the target model. When the system needs such assistance, it displays the target model to an operator along with information to help the operator determine the nature of the problem, and a satisfactory solution to it. The operator then fixes the problem, and passes control of the translation process back to the system to be completed in the background. If the system needs further assistance, it will repeat the process of notifying the operator, and displaying the problem. During the process of creating the geometry in the target model and also upon completion of translation of all the geometry, the system analyzes the quality of the translation as described above to ascertain the translation accuracy. If it finds a discrepancy or problem with either the geometry or with the process of creating the geometry, it displays the discrepancy/problem to a human operator for assistance in resolving the differences between the models.

Geometry Types

B-Rep Solid Geometry

Modern CAD systems create geometry in a number of ways. These include surfaces, curves, and solids. There are two general methods for creating and defining solid geometry. One is to create a series of boundary surfaces that enclose a volume. The CAD system can then consider the enclosed volume to be solid and have material inside or outside the solid. This type of solid is commonly called a boundary representation solid or B-Rep Solid.

Figure 20 illustrates a solid cylinder 272 created by creating B-Rep Surfaces, such as cylindrical surface 274 and end surface 276, to enclose a volume. These boundary surfaces can be created in a variety of ways including lofting, extruding, blending, sweeping, coons patch (specific mathematical method of creating a surface based upon its boundary curves), ruled surface, revolved surface, and others. Translation of these types of features is accomplished by converting the boundary surfaces along with their boundary curves to the target CAD system, enclosing the volume in the target CAD system and creating the solid.

Constructive Solid Geometry (CSG)

The other predominant method of modeling 3D solids is CSG. This involves the creation of specific geometric features and combining them to define the final solid. The individual solid features created for the CSG method may be either "parametric" features or B-Rep solid features. "Parametric" features refer to specific geometric failures created by specifying the parameters that define a primitive (a geometric object that is fully defined by its parameters). Many parametric features use a combination of geometric entities and operations to fully define the feature. Simple primitives used in parametric features include such cuboids, cylinders, spheroids, ellipsoids, toroids, etc. Examples of more complex parametric features include sketched features where the geometric object is defined by sketching a cross-section and then extruding it a specific distance, sweeping it along a trajectory curve, or revolving it about an axis. Sketched features are more complex in that they can include just about any shape that can be sketched; however; they can still be driven by parameters by dimensionally constraining the sketches, and the extrusion depth, revolve angle, or sweep trajectory. Complex geometric features can also be created by blending solid geometry between two or more sketches using suitable mathematical algorithms.

Figure 21 illustrates a solid model geometry 278 created by extruding a circle section to create a cylinder, sweeping another circle along a trajectory to add a swept feature and revolving a complex section to create a revolved feature. A first circle system is extended to form extended section 280. A second circle section (or sweep section) 282 is swept along a sweep trajectory 284. A complex section is revolved to create a revolved feature such as revolved section 286.

Translation of these types of features is accomplished by extracting the pertinent parameters, sections, trajectories, and other associated geometry from the source model, and recreating them in the target CAD system using the same or equivalent construction geometry and parameters.

Assembly Models

Solid models are usually organized into individual parts that are then assembled to other part models to create assembly models. Parts and sub-assemblies may be assembled together to form a higher-level assembly. Different CAD systems use different methods to accomplish this. One is to create a separate assembly model that

refers to the individual part or sub-assembly models file and contains assembly constraints and part (or sub-assembly) location and orientation information in the assembly. Another is to include each part (or sub-assembly) model in the assembly model as a separate entity. Some CAD systems use a combination of the two methods described above.

Figures 22A and B illustrate two methods of storing assembly model data. Translation of assembly models will require mapping of these two main types of models from one to the other when the two CAD systems use the different assembly architectures. Assembly model 288 is mapped into "Part 1" 289, "Part 2" 290, "Part 3" 291, and sub-assembly 287. Sub-assembly 287 is mapped into "Sub-Assembly Part 1" 293 and "Sub-Assembly Part 2" 295. According to another method, assembly model 288 is mapped into "Part 1" 1289, "Part 2" 1290, "Part 3" 1291, and "Sub-Assembly" 1292. "Sub-Assembly" 1292 is mapped into "Sub-Assembly Part 1" 1293 and "Sub-Assembly Part 2" 1294.

Constraints, Relations, Associations, and Solid Properties

It is common for most modern CAD software systems to assign constraints to their solid model geometry. These constraints can be geometric or dimensional for part feature geometry, and inter-part (or inter-sub-assembly) relational for assembly model geometry. In addition, relations can be established between parameters of a single feature (intra-feature relations), multiple features in the same part model (inter-feature relations), or multiple features of two or more part (or sub-assembly) models in an assembly (inter-model file relations). For example, relations can be established between features of a model in which a specific sketch plane for a feature can be related to a specific face for an existing feature, or the dimensions or parameters for a section sketch can be related to an existing feature. In this way, model behavior can be controlled in the event changes are made to the model that affect features downstream, or other parts of the assembly.

Figure 23 illustrates the relation with a part 292 between the location of a hole 296 and a revolve axis 294 of part 292. This type of information would be extracted from the source model and reconstructed in the target CAD system in a manner that is commonly used in the target CAD system.

The solids in these models can be assigned physical properties so that the "Solid Model" can now simulate various characteristics such as mass properties, static behavior, dynamic behavior, and others. These properties can be extracted from the source model and inserted into the target model as well.

Operator Tool Set Theory of Operation

Overview

The purpose of an operator toolset is to improve the efficiency of CAD model file translations by providing the operator with the information necessary to: recognize when and where the target model geometry is different from the source model geometry; complete the creation of geometric features that cannot be translated automatically by software; and allow for the conveyance of the necessary geometric information without requiring the operator to refer to the source CAD model file.

Applicant has developed a set of tools that does several things. First, the set of tools shows the operator where the surfaces and edges of the corresponding geometric features of the other CAD model are located in relation to the target model. Secondly, the set of tools details the status of each geometric feature that was converted. Thirdly, the set of tools informs the operator of the type of each geometric feature in the source model along with the methods and parameters used to construct them. Finally, the set of tools displays the required parameters and references required to complete features that were not automatically constructed in the target CAD system.

By providing the operator with this detailed information, the operator can quickly and efficiently complete CAD model file translations that duplicate as much as possible the original design intent, and are accurate.

Geometry Deviations

Point Cloud Displaying Bad Geometry

Applicant's translation system uses the Geometric Analyzer (GA) that it has developed to determine if and where two CAD model files are geometrically different. Once the differences have been determined by the GA, the bad points are displayed in the target model file. By directly viewing these representations of where the surfaces and edges of the erroneous geometric features are supposed to be, the operator can determine the nature of the discrepancy.

Figure 24 illustrates an exemplary target model 298 having bad points 300. Bad points 300 indicate an incorrectly translated fillet radius.

Surface Representations

If desired, an operator can import a Non-Uniform Rational B-Spline Surface (NURBS) from the surfaces of selected features directly from the source model file, providing the operator with points along the surfaces and edges as well as the surfaces themselves. A NURBS surface is a standard format provided by Applicant for exporting CAD surface information to enable translation and analysis. Most CAD systems possess the functionality necessary to export NURBS (or NURB surfaces).

Figure 25 illustrates a surface 302 from the source model provided on the target model 298. Accordingly, surface 302 enables an operator to visualize where an erroneous fillet surface should be on target model 298.

Analysis Report

An analysis report is displayed to the operator detailing the type of analysis that was performed, and the quantity of errors found. If the translation is completely accurate, the report summary quickly indicates to the operator that there are no discrepancies. The operator can quickly move to the next analysis, or the next job without the need to review the model, thus saving the operator inspection time.

When analysis reveals inconsistencies between the source and target models, the report indicates to the operator how many points are not within tolerance. Therefore, the operator is given an indication of how much of the model file is erroneous.

Translation Log Viewer

Applicant has developed a Log File Viewer that provides the operator with essential information including feature regeneration status, feature identification mapping between the source and target CAD systems, and partial geometry identifications. The operator can toggle the display parameters to display any combination of features including successfully completed features, failed features, and feature types. This allows the operator to quickly sort through the translated model file and get essential information on any feature desired.

In addition, the log file for each feature displays information regarding where to

find a feature's construction geometry such as the section(s), and trajectory (if applicable). This allows the operator to quickly filter to the desired information and find the desired geometry in the target model file.

Failed Feature Aids

In the event that a feature fails to be created in the target model file, then one or more of the following aids will be inserted into the model to assist the operator in interactively creating the failed feature.

Defining Parameter Indication

Certain feature types such as rounds (fillets) and drafts will have their defining parameter and reference information displayed to the operator. For example, if a fillet fails, then the associated model edges that are supposed to be filleted will be highlighted. In addition, the radius of the fillet will be displayed with a leader pointing to the specific edge or edges to which the radius is to be applied.

For example, Figure 26A illustrates a source model 297 for target model 298 (shown in Figs. 24-26). Figure 26B illustrates a failed fillet 306 which has been annotated with defining parameters and references.

Likewise, drafts will be noted by placing the neutral plan in place, and highlighting the specific surfaces to apply the draft that applies to the specific surface(s). The draft angle will be indicated with a leader to the surfaces. Similar information can be provided for chamfers, patterns, copied or translated (moved) features, and mirrored features.

Construction Geometry for Failed Features

When a feature fails in the target model, usually the construction geometry required to create the feature is created in the target model. For example, a swept cut in which the cut section is a circle and is being swept along a trajectory curve might fail to actually build the cut in the target model. However, the section and the trajectory are placed in the model in the correct locations.

Figure 27A illustrates a source model having a sweep feature 312. Figure 27B illustrates a corresponding target model 310 having a sweep section 314 and a sweep

trajectory 316. Target model 310, as illustrated, has failed to build sweep feature 312 correctly, but has placed the construction geometry into the model.

The operator merely needs to interactively complete the sweep operation selecting the section and trajectory when prompted to do so by the CAD system.

Import Surfaces of Failed Features

The surface representations that are described above in the Geometry Deviations section are very accurate representations of the individual surfaces that exist in the source model. These surfaces are automatically placed in the target model by Applicant's translation system in the event a feature fails. An operator can interactively use these surfaces to create a boundary representation (B-Rep) of the failed feature, thereby completing the feature.

The surfaces can also be used by the operator simply as an aid in determining what the specific feature should look like. If an operator decides to build the feature using parametric or other common techniques instead of a B-Rep, then the new feature can be compared to the surfaces as an alternative method of determining accuracy. The surface could also be used to assist in generating necessary construction geometry for the feature by using its edges, or important intersections with other features as guides. Therefore, the surface can be a very powerful tool for the operator to finish features that for some reason could not be translated by software alone.

TTI TO CATIA SOFTWARE DESIGN

Note: TTI refers to Applicant's intermediate file format.

1. Introduction and Background

The purpose of this document is to provide an overview of the various functions used in the TTI2CAT software system. The overall process and the key functions are described along with the CATIA functions used in these functions.

1.1 Terms and Definitions

None

1.2 Related Documents

7000	Translator Specification – Top Document
7002	Catia Supported Feature Specification

2. Description of Functions

2.1 Function Prototypes

int go1021 ()

Description: This is the main function invoked by CATIA when the user interactively starts the tti2cat application.

Inputs: None

Outputs: int success 1 – success, 0 - failure

CATIA IUA Functions Used: None

int fromtti()

Description: This is the core function that has the whole process of model creation into it. "go1021()" calls this function after getting the translation configuration. This function opens the mtf file, sets up a unit for the model, creates a default coordinate system, reads the feature information from mtf file and then based on the configuration setting makes function call to translate features and surfaces and do pixie analysis.

Inputs: None

Outputs: reuturns 0 if succesful else -1

CATIA IUA Functions Used:

gicuni(): to set unit

int create_features (FILE *err_file,
int number_of_features,
struct Feature_struct *first_feature,
char* file_path,
int elem_default_csys)

Description: this is the main function that creates the CATIA geometry. It is called by fromtti(). The flow inside the function goes as follows. We

traverse through the feature list and identify the features that are to be included in the translation from the configuration setup. Having identified the feature to be translated, we find the type of the feature and call the corresponding function. If the feature created is not the first one, a Boolean operation is applied with the existing features.

Inputs:

FILE *err_file : log file

int number_of_features : number of features

struct Feature_struct *first_feature: pointer to the first feature in the list of features

char* file_path : path for the sgf files

int elem_default_csys : handle for default coordinate system

Outputs: returns 0 if successful else -1

CATIA IUA Functions Used:

gicide():set the name of the feature

gisetl(): search for the existing solid

giwbop(): perfrom the boolean operation

int create_surfaces (int number_of_features,
struct Feature_struct *first_feature,
char* file_path,
int elem_default_csys)

Description: create_surfaces is the main function that creates the CATIA surface geometry. We traverse through the list of features, identity the ids that are to be included into the translation and call create_datum_surface to create it.

Inputs:

int number_of_features : total number of features in the model

struct Feature_struct *first_feature: first feature of the feature list

char* file_path : file path to find sgf files

int elem_default_csys : handle for default coordinate system

Outputs: returns 0 if successful or else -1

CATIA IUA Functions Used:

giccax(): setting the axis

int create_default_csys(FILE *err_file)

Description: create_default_csys creates a 3D Axis system at the default location in the model

Inputs:

FILE* err_file: log file

Outputs: int Element number of the CSYS

CATIA IUA Functions Used:

gcwaxs(): creates a new axis

int create_extrusion(FILE* err_file,
struct Feature_struct *current_feature,
int elem_default_csys)

Description: create_extrusion creates an extrusion in CATIA. The flow is almost same for all the calls concerning feature creation. Set default coordinate system, create the main cross sections, identify the geometric entities, and create all the data required for feature creation. Finally call Catia function to create the feature and perform the error check.

Inputs:

FILE *err_file	: log file
Feature_struct *current_feature	: pointer to current feature
int default_csys	: handle to default coordinate system

Outputs: returns feature id incase successful or else -1

CATIA IUA Functions Used:

giccax(): setting axes.

giride(): getting the name.

gicide(): setting the name.

gcwspm(): creating the extrusion.

int create_revolution(FILE* err_file,

struct Feature_struct *current_feature,
int elem_default_csys)

Description: create_revolution creates a revolution in CATIA. The flow is almost same for all the calls concerning feature creation. Set default coordinate system, create the main cross sections, identify the geometric entities, and create all the data required for feature creation. Finally call Catia function to create the feature and perform the error check.

Inputs:

FILE *err_file : log file
Feature_struct *current_feature : pointer to current feature
int default_csys : handle to default coordinate system

Outputs: returns feature id incases successful or else -1

CATIA IUA Functions Used:

giccax(): setting axes.
giride(): getting the name.
gicide(): setting the name.
gcwsva(): create revolution

int create_swept_solid(FILE* err_file,
struct Feature_struct *current_feature,
int elem_default_csys)

Description: create_revolution creates a revolution in CATIA. The flow is almost same for all the calls concerning feature creation. Set default coordinate system, create the main cross sections, identify the geometric entities, and create all the data required for feature creation. Then, finally call Catia function to create the feature and perform the error check.

Note that create_swept_solid has a minor differences when compared to extrude or revolution. In swept_solid, two sections are created: one for the main section and other for the trajectory. However, the sequence doesn't get altered.

Inputs:

FILE *err_file : log file
Feature_struct *current_feature : pointer to current feature

int default_csys : handle to default coordinate system

Outputs: returns feature id incases successful or else -1

CATIA IUA Functions Used:

giccax(): setting axes.

giride(): getting the name.

gicide(): setting the name.

gcwswe(): create the swept solid.

int create_unsupported_feature(FILE* err_file,
struct Feature_struct *current_feature,
int elem_default_csys)

Description: create_unsupported_feature creates an unsupported_feature in CATIA. The flow is more or less same as extrusion or revolution except for the fact that no Catia call is made to create any feature.

Inputs:

FILE *err_file : log file

Feature_struct *current_feature : pointer to current feature

int default_csys : handle to default coordinate system

Outputs: returns feature id incases successful or else -1

CATIA IUA Functions Used:

giccax(): setting axes.

giride(): getting the name.

gicide(): setting the name.

int create_cylinder(FILE* err_file,
struct Feature_struct *current_feature,
int elem_default_csys)

Description: creates a cylinder. The flow of the function is as follows. Default system is set, required data is created, then Catia function is called to create the cylinder followed by error check.

Input:

IFILE *err_file : log file
Feature_struct *current_feature : pointer to current feature
int default_csys : handle to default coordinate system

Outputs: returns feature id incases successful or else -1
CATIA IUA Functions Used:
giccax(): setting axes.
gcwscr(): create the cylinder.

int create_datum_surface(FILE* err_file,
struct Feature_struct *current_feature,
int elem_default_csys,
char* path)
Description: create_datum_surface creates a level 2 surface in CATIA. The flow of function goes as follows. From the sgf files, the NURBS data corresponding to that surface are read. A function call is made to create surface from NURBS. Errors are checked at the end.
Inputs:
IFILE *err_file : log file
Feature_struct *current_feature : pointer to current feature
int default_csys : handle to default coordinate system
char* path : path for sgf file(has surface information)
Outputs: 0 if successful else -1
CATIA IUA Functions Used:
giccax(): setting axes.
giclay():Set the layer

int create_round(FILE* err_file,
struct Feature_struct *current_feature,
int elem_default_csys)
Description: Under development stages
Inputs:
Outputs:

CATIA IUA Functions Used:

int create_section(FILE *err_file,
struct Section_struct section,
int* section_elements,
int *section_axis,
int *section_plane)

Description: create_section creates a Section and all the elements of the section.

Inputs:

FILE* err_file	: log file
struct Section_struct section	:
int* section_elements	:
int *section_axis	:Address for Element id for the axis system created at the origin of the section
int *section_plane	:Address for Element id for the plane created for the section plane

Outputs:

Int * section_elements	:List of Elements of the section
int *section_axis	:Element id for the axis system created at the origin of the section
int *section_plane	:Element id for the plane created for the section plane

CATIA IUA Functions Used:

giccax(): setting axes.

giclay():Set the layer

int create_entities (FILE *err_file,
int number_of_entities,
enum Entity_Type *entity_type,
int *entity_id,
union Entity_struct *entities,

int* section_elements)

Description: create_entities creates all the entities in a section. Currently supported entities are Line, Circle, Arc and Spline.

Inputs:

int number_of_entities	: number of entities in the list
enum Entity_Type *entity_type	: type of the entity
int *entity_id	: id
union Entity_struct *entities	: list of entities

Outputs:

int* section_elements	:List of Element ids of the section as returned bt catia

CATIA IUA Functions Used:

giccax(): setting axes.

giclay():Set the layer

```
int create_line (FILE* err_file,
                 struct Line_struct my_line,
                 int line_id)
```

Description: creates a 3D-line

Inputs:

FILE* err_file	: log file
struct Line_struct my_line	: structure of my line
int line_id	: id

Outputs: Element id for the line

CATIA IUA Functions Used:

gcwln(): create line

gictxt(): modify line type

```
int create_circle (FILE* err_file,
                   struct Circle_struct my_circle,
                   int circle_id)
```

Description: creates a 3D circle

Inputs:

```
                FILE* err_file                  : log file
                struct Circle_struct my_circle  : structure holding circle details
                int circle_id                   : id
                Outputs:    Element id for the circle
                CATIA IUA Functions Used:
                gcwcir(): creates circle

        int create_arc (FILE* err_file,
                        struct Arc_struct my_arc,
                        int arc_id)
                Description: create_arc creates a 3D arc
                Inputs:
                FILE* err_file                  : log file
                struct Arc_struct my_arc        : structure holding arc details
                int arc_id                      : id
                Outputs:    Element id for the arc
                CATIA IUA Functions Used:
                gcwcir(): creates arc

int create_spline (FILE* err_file,
                        struct Spline_struct my_spline,
                        int spline_id)
                Description:
                FILE* err_file                  : log file
                struct Spline_struct my_spline  : structure holding spline
                                                  details
                int spline_id                   : id
                Outputs:    Element id for the spline
        CATIA IUA Functions Used:
                gscsm1(): create spline

        int pixiedust(int elem_default_csys)
                Description: pixiedust is the main pixiedust program called from tti. This
```

function, in turn calls functions for pixie analysis for tessellated points, surface points and edge points.

Inputs:

int elem_default_csys : handle to default coordinate system.

Outputs: None

CATIA IUA Functions Used: None

int pixiedust_tessellated(int elem_default_csys)

Description: pixiedust_tessellated is the pixiedust program for analyzing tessellated points. It reads the pixie point information from the pixie file or the bad pixie file depending on whether it's a reanalysis or not. It then creates these points and calls analyze_tessellated for running the analysis.

Inputs:

int elem_default_csys : handle to default coordinate system.

Outputs: returns 0 if successful else -1

CATIA IUA Functions Used:

giwpt(): create points

giccax(): set axes

int analyze_tessellated (FILE* err_file,
FILE* bad_pxi_file,
int number_of_points,
int *pointid,
point_struct* input_points,
float tolerance)

Description: analyze_tessellated gets all solid elements and checks each point for a distance from that element. To do this, it takes the points one by one and projects them on the surface of the solid. Then it compares the distance between the point formed on the solid surface to the actual point. If the distance is more than the tolerance then it writes that pixie data to bad pixie file. It is used for both, tessellated data analysis and surface analysis.

Inputs:

FILE* err_file : log file

FILE* bad_pxi_file : file pointer to the file where bad pixie points are written.

int number_of_points : number of pixies points in the list

int *pointid : their ids

point_struct* input_points : list of points

float tolerance : tolerance for testing

Outputs: if successful returns 0 else -1

CATIA IUA Functions Used:

gsopob(): project the point on the solid.

gusrea(): get the point from the stack.

girmat(): get XYZ from the projected point.

gieras(): erase point.

int get_exact_solids (FILE *err_file,
int* solids_list)

Description: get_exact_solids is the function that gets all the exact solids. It uses Catia functions to search through the current workspace and identify all the exact solids and return them in the form of the list. This function is called by analyze_tessellated for getting all solid elements.

Inputs:

FILE *err_file : log file

Outputs: returns the number of solids

int* solids_list : list of solids

CATIA IUA Functions Used:

gisetl():search for solids

giride():get name

int pixiedust_surface(int elem_default_csys)

Description: pixiedust_surface is the pixiedust program for analyzing surface points. It reads the pixie points information from the surface pixie file or the bad surface pixie file depending on whether it's a reanalysis or

not. It then creates these points and calls analyze_tessellated for running the analysis.

Inputs:

int elem_default_csys : handle for default coordinate system.

Outputs: returns 0 if successful else false

CATIA IUA Functions Used:

giccax(): sets axes

giwpt(): create points

int pixiedust_edge(int elem_default_csys)

Description: pixiedust_edge is the pixiedust program for analyzing edge points. It reads the pixie points information from the edge pixie file or the bad edge pixie file depending on whether it's a reanalysis or not. It then creates these points and calls analyze_edge for running the analysis.

Inputs:

int elem_default_csys : handle to default coordinate system

Outputs: returns 0 incase of success or else -1

CATIA IUA Functions Used:

giccax(): sets axes

giwpt(): create points

int analyze_edge (FILE* err_file,
FILE* bad_pxi_file,
int number_of_points,
int *pointid,
point_struct* input_points,
float tolerance)

Description: analyze_edge gets all edge elements and checks each point for a distance from that element. Since the process of edge analysis is time consuming, the test has been divided into several levels.

Inputs:

FILE* err_file : log file

FILE* bad_pxi_file : bad edge pixies are written to these

files

int number_of_points : number of points in the list

int *pointid : their ids

point_struct* input_points : list of points

float tolerance : tolerance for edge analysis

Outputs: returns 0 if successful else -1

CATIA IUA Functions Used:

gsopoe(): project the point on the edge

girmat(): get edge from the stack

gieras(): erase the edge

int get_edges (FILE *err_file,
int* edge_list,
int* solid_of_edges,
Bounding_box *edgebox,
double tolerance)

Description: get_edges is the function that gets all the edges of the exact solids

Inputs:

FILE *err_file : log file

double tolerance : tolerance

Outputs: returns number of edges if successful else -1

int* edge_list : list of edges

int* solid_of_edges : id of the solid to which the edge belongs

Bounding_box *edgebox : bounding box around the edge

CATIA IUA Functions Used:

gcwvfr():create a volume of the exact solid

girnli(): find out the faces in the volume

gislim():get the ids

gsoioo(): check to see if points are identical

gslimp():get the end points of both the edges

gsdiee(): check for distaance.

gieras(): erase the points.

Bounding_box get_edge_bounding_box(int edge_id)

Description: get_edge_bounding_box gets the bounding box for a CATIA edge

Inputs:

int edge_id : id of the edge

Outputs: returns a bounding box surrounding the edge

CATIA IUA Functions Used:

gseoed(): create a curve

giresi(): get the length of the curve

girmat():get the discription block

gieras(): delete the curve

check_inside_bounding_box (point_struct input_point,
Bounding_box box)

Description: check_inside_bounding_box checks to see if a point is inside a bounding box.

Inputs:

point_struct input_point : point to be checked

Bounding_box box : bounding box

Outputs: returns 1 if inside else 0

CATIA IUA Functions Used: None

2.2 Descriptions of Functions

Over all Flow of Program

See Figures 28 and 29.

2.3 Languages

The GUI module will be implemented using the C++ programming language and the FLTK Toolkit. The Translation Analyzer will be implemented using the C++ programming language and the Open GL Optimizer. The Surface Generator modules

will be implemented using the programming language supported by each individual API.

2.4 Operating System(s)

Windows NT and Windows 95/98 will be the initial supported platforms.

2/5 Databse/Knowledge Base

TBD

Import/Export of Surface Data

Overview

Most CAD translators use surface files to translate geometry across from one modeling system to another. TTI translates surfaces only when it is the only type of data which can be successfully translated. This will occur when:

- The feature in the source CAD system is a surface.
- The feature in the source CAD system is an imported feature.
- The feature fails to be created in the target CAD system.
- or The customer has requested a surface mode.

Although a surface CAD to CAD translation is not generally the most desired type of translation, surface information about the model is important to TTI for purposes of Level 4 (associativity) translations and use in analysis tools.

Surface information is kept by the CAD system in any of a number of various ways. Most CAD systems, however, have the capability of converting any of these surfaces to NURB surfaces NURB surfaces are able to exactly represent any surface. This means that there is no approximation of the surface which is present in the source CAD system.

The NURB surface is the TTI standard format for exporting of CAD surface information for the means of translation and analysis. Please see the Surface Geometry File Format in *TTI File Format Standards*, Document ########.

Although most CAD systems possess the functionality to export NURB surfaces, most designers are not familiar enough with NURB surfaces to effectively or efficiently modify NURB surfaces. Therefore, when TTI translates surface models, maximum effort is placed on creating the surface in the target CAD system in a way that is both consistent with how surfaces are normally be created in the specific CAD system and

accurate representations of the original surface. In an effort to provide native surfaces in some CAD systems, a second type of information is supported by TTI.

The other method for storing surface information used by TTI is contour curve generation. Contour curves are generated by stepping along a parametric surface in both the u and v parameter directions and writing points out along these contours. Then curves are fit through the contour points to create the contour curves. Please see the U and V Curve File Format in *TTI File Format Standards*, Document #########.

Exporting NURB Surfaces

The Exporting of NURBS surfaces can be as easy as a couple of function calls to CAD system API utilities. However, some CAD systems require more effort to convert native surfaces to NURB surfaces for export. This section will explain the special steps for surface conversion to NURBS for each CAD system currently supported by TTI.

1. Catia

Catia maintains surfaces for each volume in the model. A hierarchy of parent-children relationships must be understood before attempting to export surface information from Catia. There are two types of solids in Catia, Exact and Mock-up(Dumb). An exact solid is created by combining primitives such as prisms, cylinders, fillets, etc. into a solid model. An exact solid does not naturally have a volume associated with it. The second type of solid, a dumb solid, is created using B-rep elements. The hierarchy of information that must come together to create a dumb solid is shown in Figure 30.

Applicant exports surfaces related to both dumb solids and exact solids. Therefore, in order to create the surfaces associated with the exact solids in a Catia model, a volume is first extracted from every exact solid in the model. The extraction of a volume associated with an exact solid forces the creation of faces and surfaces. If there are no exact solids in the Catia model, then all the surfaces should already be present in the model and no volume creation is needed.

Catia has two types of surface information, polynomial and planer. Using Catia API function calls gslnsf and gssnsf, NURB surface information can be directly extracted from any polynomial surface. All control point weights for a NURB surface created from a Catia polynomial surface are equal to 1. Planer surfaces in Catia are not

converted to NURBS using Catia API function calls. This is because planes are infinitely large. The control points for a NURB surface meant to represent a plane are set at the corner points associated with whatever value has been assigned to infinity in Catia. Therefore, the following methodology was developed to export NURB surface information from planer surfaces in Catia.

1. Use the edge information to determine the planer bounding box for the trimmed planer surface.

2. Get the control points associated with the NURB surface Catia would build using (defined) infinity as the control point locations.

3. Use a shortest vector method for determining the order in which the four calculated control points (bounding box) should be reported. The theory behind this method is that the calculated control points should be in the same order as the infinite plane control points. (See Figure 31.)

Since this is a planer surface, there will be two knots at each parameter minimum and maximum value. This completes getting the NURBS information from both polynomial and planer surfaces from Catia.

The second type of surface information that is extracted by TTI is contour curve information. In order to get contour curves from Catia the number of patches which make up the surface. Then step along the u and v directions of the entire surface and evaluate the correct surface patch to get continuous U and V curve information from the surfaces. See Figure 32.

A second, more efficient way of creating contour curves is by stepping along and evaluating the NURB surface that was created to represent the CAD surface earlier. By doing this, it is possible to have one generic algorithm for creating contour curves from any surface written to the TTI SGF standard and is not dependant on the source CAD system or format in which it keeps the surfaces.

2. Pro/Engineer

Pro/Engineer keeps track of surface data in many ways, Cylindrical, Planer, Polynomial, Blended, etc. All of the Pro/Engineer surface types can be converted to NURBS through API function "ProSurfaceToNURB()."

Contour curves coming out of Pro/Engineer are created by evaluating the created NURB surface along the U and V directions.

Importing NURB Surfaces

Creating NURB surfaces in most CAD systems is not difficult. CAD users however are not usually accustomed to working with NURB surfaces. TTI, therefore will create surfaces in the target CAD system that are consistent with what is normal in that system.

1. Catia

Most Catia surfaces are polynomial surfaces. Unlike the NURB surface, a polynomial surface is not guaranteed to be exactly the same as the surface in the source CAD system. The polynomial surface, however, is more representative of surfaces normally encountered by CAD operators using Catia. Therefore, whenever the polynomial surface is within tolerance of the exact surface, TTI translations will use the polynomial representation of the surface. Otherwise, the NURB surface, is used. The overall procedure for importing surfaces into Catia is:

1. Read the NURB surface data from the SGF.
2. Create the NURB surface using Catia Free Form Design function call "gcwnbs()".
3. Use Catia Free Form Design function call "gssnb1()" to convert the NURB to a polynomial.

2. PRO/Engineer

Pro/Engineer is capable of creating a NURB surface from the TTI standard SGF format. However, these surfaces are reported on the feature tree to be "imported features". Even though these surfaces are listed as imported features, they are fully modifiable in the Pro/Engineer environment. Because "imported features" are not desirable, TTI uses contour curve information to create a Pro/Engineer native, blended surface (*TTI File Format Standards*, Document ########). Blended surfaces are not guaranteed to be exactly the same as the surface in the source CAD system, but the accuracy can be improved when needed by the insertion a higher resolution of contour curves wherever necessary. If TTI is unable to force the blended surface to be within tolerance, the NURB surface will be imported and used. Importing blended surfaces from TTI contour curve files is a two-step process.

1. Create temporary Pro/Engineer IBL files. IBL files contain contour information in a format which is directly readable by Pro/Engineer.

2. Use a macro routine to read the IBL file and create the blended surface.

(See Figure 33.)

Trimming of Surfaces

The trimming of the surfaces in the CAD system is a monumental task both interactively and internally. Due to the technically difficult nature of trimming surfaces. Some surface trimming in the target CAD system is presently performed manually.

Interactive Trimming of Surfaces

There are two basic ways to trim surfaces interactively in most CAD systems. The two main methods are:

1. Surface to Surface intersection.

2. Trim Curve on Surface.

Surface to Surface intersection is perhaps the easiest way to trim surfaces interactively. A step by step procedure is shown in Figure 34.

Interactive trimming using the intersecting surface method is the fail-safe way to trim surfaces. Using this method minimizes the possibility of gaps between the surfaces because the trimming of the two surfaces was done by the CAD system and the same curve was used for both surfaces.

The second interactive method for trimming surfaces is the use of datum curves. The curve, either imported or created interactively will have to be projected onto the surface that it is meant to trim in order to insure that the curve is on the surface at all points along the curve. This is a requirement for trimming. The curve can then be used for trimming the surface. The disadvantages to this type of trimming is the fact that small errors force the projection of the curve onto the surface. This introduces the possibility that small gaps will be introduced where surfaces are meant to intersect. This is not a catastrophic event. If there are small gaps between the surfaces, the surfaces can be modified since TTI translated surfaces are native surfaces in the target CAD system and therefore completely modifiable.

Automated Trimming

All CAD systems perform surface trimming using curves that are defined in the u-v parameterization of the surface, which is to be trimmed. This information is not easy to extract when the original surface is being converted into a NURB surface. The problem arises when the parameterization of the surface is not preserved in the conversion to NURB (see Figure 35).

The fact that there is no guaranteed mapping of the parameterization of the original surface and the NURB surface exported by TTI means that the trim information given by the CAD system, in the parameterization of the original surface, can not be applied to the NURB to trim the surface.

TTI has developed algorithms to convert 3D coordinates (x,y,z) into the associated (u,v) coordinates of the NURB surface, if such a (u,v) coordinate exists (*i.e.*, the x,y,z point is on the surface). Using these algorithms (see *TTI NURB Utility*, Document #########), TTI exports the trim information as a series of points on each trim contour/edge in the parameterization of the NURB. This trim information can be used to create a curve that will lie on the surface that is to be trimmed. This method, however, is similar to the interactive trim curve method and introduces the possibility of small gaps between surfaces due to the point array representation of the trim curves. This method of automated trimming only works in target CAD systems that provide functionality for the automated trimming of surfaces.

Mapping of Boolean Trees Between CATIA and Pro/E

CATIA and Pro/E have different ways in which they store the feature information and their associated operation. CATIA supports the creation of multi-branched boolean trees when solids are combined to form a part. In Pro/E, the regeneration of solid features is done sequentially following a single sequence. It is not possible to create a complex boolean tree and then perform a boolean operation of that tree with another complex tree. This document describes how the boolean trees are converted from one type to the other. This document also describes the data structures and utility functions created to support this conversion.

Example for formation of CATIA TREE from an Array

Given Example List

12 12345 13456
10 12343 23455
18 -1 37557
04 12309 12835
09 -4 -2
03 -5 -3

First Number: The operation (addition/subtraction!..)

Second and Third Number: if negative - represents the line number, if positive - element id.

Steps in creation of the Catia Tree:

Step 1: Create two leaf nodes and a parent node and set parent as root node.

Tree: See Figure 36.

Step 2: Since no correlation between the new formed parent node and tree, the newly formed parent goes into the list.

Tree: See Figure 37.

Step 3: Now the Tree Root is the left child of the newly created parent. List remains unchanged.

Tree: See Figure 38.

Step 4: An unassociated parent with two leaf nodes would be created and added to the list. Tree will remain unchanged.

List: See Figure 39.

Step 5: Now the root of tree is the left child and list has the right child. The Node with id -2 would be removed from the list and added to the tree. So in this case list would become empty.

Tree: See Figures 40 and 41.

Step 6: Now root of the tree is the left node and the only element is the list the

right node of the parent node with operation 03 and id -6. List, thus, would become empty.

Tree: See Figures 42 and 43.

PROE TREE:

Proe tree has a single level of hierarchy. Therefore it is more like a list. Another important difference between the two trees is that Proe tree has only unions and subtractions (operation 03). The geometric elements with similar elements (as represented in CATIA TREE) would be represented as shown in Figures 44 and 45.

The tree can be represented as a list with element number and operation. The first two nodes will have the operation set to zero.

GOING FROM CATIA TO PROE TREE:

In order to go from CATIA to proe, we need to flatten the Catia tree to single level of hierarchy. Essentially we need to extract the primitive elements in the order that they have been created along with the final operation that need to be performed. The final operation can be only union or subtraction as these are the only Boolean operations proe supports.

In order to get the primitive elements, following function is to be used.

"getListOfAllLeaves(tree, list)"

This function takes the catia tree and returns the list of all the primitive elements; *i.e.,* the leaf nodes of the element from left to right. The function forms the list by traversing from left mode node to the right most node of the tree. Hence it restores the sequence in with the primitives where created.

Along with the primitives, as discussed earlier, we also need the operation in order to flatten the tree. Following function is provided for doing so:

"List * getFinalOperationList(* tree)"

This function returns the final operation list. In order to final out the final operation list, we traverse from the leaf node up the hierarchy along its parents one by one. Only in case we come across a right child we subtraction, that we consider it as subtraction. This is because the child on the right is subtracted from the child in the left. Left child does undergo any subtraction. Also, subtraction after any subtraction down the hierarchy becomes an addition.

Also, CATIA has an operation called transformation that is not supported. The transformation is essentially a copy of current model with different position and orientation. In geometry wise it is the just the replication of the left child of the operation. In order to go from catia to proe, all these transformations have to be removed. In order to do this following function has been provided.

"List * copyLeftToRight(tree, parent)"

The function makes the replica of the left child at the right side with new element numbers. It then returns the old element and corresponding new element number so as the user could keep track of the elements on which the transformation is to be applied.

GOING FROM PROE TO CATIA

Most important thing to remember in going from proe to catia is to flatten the single level tree as much as possible. That reduces the updating time. By flattening the tree we mean that to be able to add as many branches as possible so that the tree grows in all possible directions. There are of course some obvious constrains that need to be taken care of. The sequence in which the parts have been created should to be maintained. And the operation on the part should be appropriately taken care of.

The implication of the first constrain is that the newly added node has to be added to the right of the previous node. At no point can a new node be on the left side of any of the previous nodes.

In order to take care of second constrain we grow the branches on in case when we have repeated similar number of operations.

In order to perform the above operations, following operation has been added.

```
//takes the PROE list of nodes and return a CATIA flattened tree
struct Node * getCatiaTree(struct List * list);

struct Node * return root of CATIA Tree
List * list                PROE feature and operation list.

//compare the length of left node and right node
int compareNode(struct Node * tree);

int return                 0 in case there is only a leaf node or if the left node
                           and right node are balanced.
                           1 in case number of leaf nodes in left side is greater
                           than on the right side.
                           -1 in case right side leaf nodes are more.
```

//to add a node to CATIA tree while going from PROE to CATIA
void addToCatiaTree(struct Node ** currentNode, struct Node * newNode, struct Node * newParent, int currentOperation);

currentNode	is the right side of the root node.
newNode	the new node to be added
newParent	the parent of the new node
currentOperation	to know whether there is any change in the operation

The process of creation can be understood as follows:

In "getCatiaTree()" we pass the PROE list of nodes containing the feature id and operation. In this function as we traverse through the list, we create a new node and its parent according to the information we get from the list and call the function "addToCatiaTree()".

In "addToCatiaTree()" we check whether the new operation matches with the last operation or not. In case if it matches we know that branching has to be done. In case if it doesn't we make current root node the left of the new parent and the right node the right of the new parent. The new parent then is set to be the root of the CATIA tree.

In case we have to branch we check whether the right of the root node (in our case it is passed as the current node to the function), whether it is balanced or not (the case includes whether it's a leaf node or not). In case it is balanced, we shift the current node to as the left child of the new parent and the new node the right child. The we make the new parent the right child of the previous parent of the current node. In case if it is not balanced, we traverse down the right child until we find a balanced child. Then we perform the same operation.

The result is that we get a balanced grown tree.

The growth of the tree can be understood from following figures:

Feature id from PROE: 12345 operation: None (as it is base feature)
(See Figure 46.)

Feature id from PROE: 12346 operation: UNION
(See Figure 47.)

Feature id from PROE: 12347 operation: SUBTRACTION
(See Figure 48.)

Feature id from PROE: 12348 SUBTRACTION

12345 12346 12347 12348

Feature id from PROE: 12349 operation: SUBTRACTION

(See Figure 49.)

Steps of Integration:

1. Pass the array to the following function and get a pointer to the root of the tree.

```
struct Node * formTreeFromArray(int * array, int lengthOfArray);
```

2. Cleaning of the Transformation
   a. get all the leaf nodes by making the following function call

```
void getListOfAllLeaves(struct Node * tree, struct List ** list);
```

   b. Traverse through the list and see whichever has parent as transformation operation. Only when the child is the right child of that parent, then we need to proceed.

```
for ( i = 1; i <= getListLength(list); ++i) {
        if ( getItem(list,i) != NULL) {
                node = getItem(list,i)->data;
                if ( node ! NULL) {
                        //do ur job
                }
        }
}
```

   c. copy the left side to the right side and assign the new element numbers. Following function does that.

```
list = copyLeftToRight(tree, parent);
```

   d. Now the new elements have been added to the list. These changes have to be incorporated in to the actual data structure. That is copy of each element has to be made and then required transformation has to be applied. Make sure the new elements have the same number corresponding to the one in the Tree. New element numbers and their corresponding old element number can be accessed from the list. They are stored one after another (old element number followed by the new element number).

3. Once all the transformations are removed.., list of all leaf nodes can be accessed using

getListOfAllLeaves(tree, &list);

4. and the operations for the all the leaf nodes from

listOfInt = getFinalOperationList(tree);

5. once done tree can be deleted as follows

deleteTree(&tree);

The relevant data structures with their associated function and usage:

LIST

```
struct List {
        struct List * next;
        struct Node * data;
        int id;
};

//to add item to the list
int addItem(struct List ** list, struct List * item);

//to delete all the items from the list
int deleteAll(struct List ** list);
//to delete item based on index
int deleteItem(struct List ** list, int index);

//to search an item based on Id
int searchList(struct List * list, int id);

//get item knowing the index
struct List * getItem(struct List * list, int index);

//get the list length
int getListLength(struct List * list);

//delete item based on Id
int deleteItemBasedOnId(struct List ** list, int id);
```

In all, the above functions all users to add item to the list, delete, access them and get the length of the list.

Usage:

```
struct List * list;
struct List * listItem;
```

```
struct Node * node;
```

1. Adding a list item

```
listItem = (struct List *)malloc( sizeof(struct List));
listItem->next = NULL;
listItem->data node;
listItem->id = id;
addItem(&list, listItem);
```

2. Deleting all or based on id

```
//deleting based on id
deleteItem BasedOnId (&list, Id);
//deleting all
deleteAll(&list);
```

3. Accessing items from the list

```
for ( i = 1; i <= getListLength(list); ++i) {
        if( getItem(list,i) ! NULL) {
                node = getItem(list,i)->data;
                if( node != NULL) {
                printf("%d element %d\n", i, node->element);
        }
}
```

ListOf Int

```
struct ListOftnt {
        struct ListOfInt * next;
        int intValue;
};

//add the list of int
int addIntItem(struct ListOfInt ** list, struct ListOfInt * item);

//delete all the items from the list of int
int deleteAllIntItem(struct ListOfInt ** list);

//get item knowing the index
struct ListOfInt * getIntItem(struct ListOfInt * list, int index);

//to get the length
int getIntListLength(struct ListOfInt * list);
```

ListOfInt functions are pretty much just the modification of the functionalities provided for List. This list is returned for the list of operations.

TREE

```
struct Node{
        struct Node * left;
        struct Node * right;
        struct Node * parent;

        int  id;              //line number
        int  operation;       //operation.., last leaf nodes will have -1
        int  element; //element number
};

//to add left node to the parent
int addLeftNode(struct Node * parent, struct Node * leftNode);

//to add right node to the binary tree
int addRightNode(struct Node * parent, struct Node * rightNode);

//create a node with given values
struct Node * createNode(int operation, int element, int id);

//form the tree with the array
struct Node * formTreeFromArray(int * array, int lengthOfArray);
```

Note: array to be given expects following format:
Operation, leftNode element number or line number, right node element number of line number, operation...and so on.

```
//search the tree based on the id
struct Node * search(struct Node * tree, int id);

//clean up the tree
void deleteTree(struct Node ** tree);

//get all the list of all leaf nodes from left to right
void getListOfAllLeaves(struct Node * tree, struct List ** list);
```

//replicates the left side of the parent to the right side with newly assigned element number and returns the list of old nodes and new nodes

```
struct List * copyLeftToRight( struct Node * tree, struct Node * parent);

//recursive function used by copylefttoright in order to replicate the tree recursively
void replicateTree(struct Node * parent, struct Node * nodeWhereTobeCopied, struct
List ** list);
```

//return the list of integer containing the final operation list

```
struct ListOfInt * getFinalOperationList(struct Node * tree);
```

The user would use above functions for following purpose:

```
struct Tree * tree;
struct List * list;
struct ListOfInt * listOfInt;

//array would come from catia history tree.
tree = formTreeFromArray(array, lengthOfArray);

//after creating the tree copyLeftToRight could be used to remove all the transformation
//returned list would contain the old node and new old
list = copyLeftToRight(tree, parent);

//once all the transformations are removed.., list of all leaf nodes can be accessed using getListOfAllLeaves(tree, &list);

//and the operations for all the leaf nodes from
listOfInt = getFinalOperationList(tree);

//once done tree can be deleted as follows
deleteTree(&tree);
```

Round Edge Finding Utility

Overview

The translation process for rounds is more complicated than the translation process for basic features such as extrusions, revolutions and most primitives because it is a feature that is referenced to a feature (edge) that is destroyed when the round is created. Because of this an algorithm was developed for determining where the reference edge(s) were located before the round feature was created.

Round Feature

The round feature TTI MTF format is shown in Figure 50.

The figure above shows the edge referenced by the round feature as a dashed line with four points located along it. This is how the TTI translation process locates the edge on which to create the round in the target CAD system. When the source CAD system generates the MTF file the entry for the round does not include the edge information. The entry created by the source CAD system is shown below.

```
FEATURE 30
        NUM_SECTIONS        0
        NUM_SURFACES        1
        SURFACE_IDS 31
        BOOLEAN             UNION
        FEATURE_TYPE        ROUND
        NUMBER_OF_RADII     1
        RADII 6.000000
        NUMBER_OF_EDGES     0
        COMMENT
```

Utility Application in the Translation Pipeline

After all features have been written to the TTI MTF format, a utility application reads the file and searches for rounds. When a round is found the application will read the surface or surfaces associated with the round. Using the surface information and radius information four points are placed where the original edge would have been. A diagram of the translation process with the round utility application is shown in Figure 51.

Edge Locating Algorithm

For Rounds with Constant Radius, the Edge can be found by extending the tangents of the round cross-section until there is an intersection. The intersection point will be on the edge referenced by the round (see Figure 52).

The cross-section of the round is extracted from the NURB surface associated with the round. In order to do this, the radial and axial directions need to be associated with the u and v parameters of the NURB surface. If the NURB surface was created with chord length parameterization, the magnitude of the second derivative will equal the ratio of radius/chord length.

ASSUMPTION: **The NURB surface associated with the round was created using chord length parameterization.**

If this assumption is false, a new relationship needs to be developed for centripetal parameterization.

After the radial direction has been identified, the dot product of the tangent vector at one end and the vector along the chord is calculated giving the cosign of the angle between the two vectors (cos(a)). The edge point will then be a point offset along the tangent vector with a magnitude of (chord length)/(2*cos(a)). See Figure 53.

After the first point has been calculated, the value of the parameter associated with the axial direction of the round is incremented by (1/3)(max - min) and another edge point is calculated. This is repeated until the parameter value reaches its maximum. See Figure 54.

These four points are added to the data structure and the number of edges is increased by one. This whole process is repeated for every surface associated with the rounds. Therefore:

Number of Edges = Number of Surfaces

The final output for a round with one edge is shown in the figure below.

```
FEATURE  30
         NUM_SECTIONS          0
         NUM_SURFACES          1
         SURFACE_IDS 31
         BOOLEAN               UNION
         FEATURE_TYPE          ROUND
         NUMBER_OF_RADII       1
         RADII 6.000000
         NUMBER_OF_EDGES       1
         POINT     0.000000    0.000000    0.000000
         POINT     1.000000    0.000000    0.000000
         POINT     2.000000    0.000000    0.000000
         POINT     3.000000    0.000000    0.000000
         COMMENT
```

Section Quality Assurance

Overview

CAD systems do not treat 2D cross-section entities the same. Because of this, some CAD systems are more tolerant of how a cross-section is created. In order to translate features from one system to another, cross-sections need to be reduced into a basic form and structure that all CAD systems will allow. There are three issues involved in section integrity:

1. Open vs. Closed Sections
2. Entity Connectivity or "Entity Chaining"
3. Disjointed Sections

The following document explains the methods and algorithms used by Translation Technologies Incorporated to recognize and fix sections that are not friendly to all CAD systems.

Section Integrity Utility in Translation Pipeline

Algorithms contained in the TTI MTF file export utility perform all section corrections. This insures that all sections existing in the MTF and SDF file formats are in a state which is interpretable by all TTI supported CAD systems. See Figure 55.

1. Open vs. Closed Sections

Open Section: A section for which the combination of all entities does not form a complete loop. See Figure 56.

Closed Section: A section for which the combination of all entities forms a complete loop. See Figure 57.

Identifying an Open Section

The toolkit contains functionality to identify all open sections. The process in identifying an open section includes:

- Obtaining a list of all entities in the section
- For each entity find the endpoints
- For each endpoint find another end point that matches it
- If there is no matching endpoint then the section is open.

There are a few special cases. Some entities are able to close themselves. These entities are:

1. Circle
2. Arc
3. Spline

TTI treats all entities as if they have two endpoints. Therefore, when an entity closes itself, the two endpoints are the same and the section is properly identified as closed. See Figure 58.

Closing of Open Sections

Some CAD systems allow a user to specify an open section for some features. Other CAD systems prefer to have all sections closed. In order to deal with the most common open section, TTI's toolkit is able to close open sections on revolved features. The process for closing open sections associated with revolved features is:

1. Identify the open section. See Figure 59.
2. Identify the unmatched endpoints. See Figure 60.
3. Identify the centerline. See Figure 61.
4. Construct lines through the unmatched endpoints and perpendicular to the centerline. See Figure 62.
5. Determine where these new line segments cross the centerline. See Figure 63.
6. Construct a line segment through the intersection points along the centerline to completely close the section. See Figure 64.

Note: The line segment constructed in part 6 is ignored by some CAD systems.

2. Entity Connectivity or "Entity Chaining"

CAD systems require that all entities in a section be in the correct order. Other CAD systems only require that all entities be defined, they are not required to be in any given order. In order to support as many CAD systems as possible, TTI formats require that the entities of a section are placed in the proper order. The TTI toolkit will take sections which have non chained entities and chain them before writing them to the TTI file format. The process of doing this involves:

1. Identify all the entities of the section.
2. Obtain the end points for all the entities.
3. Start with one entity and endpoint.
4. Find another entity with the same endpoint.
5. Use the new entity and new endpoint to find a match.
6. Repeat 4 and 5 until all entities have been used, or section has been closed.

Note 1: If at any point the section is determined not to be closed, an error is returned and the entities will not be chained.

Note 2: Construction or reference entities are not used for entity chaining.

3. Disjointed Sections

The third variation in 2D cross-section definitions between CAD systems is the support for disjointed sections. Disjointed 2D cross-sections will allow a user to create multiple solids or cuts(not contained by a solid) as a single feature if the CAD

system supports it. Many of the CAD systems do not support this type of feature generation.

Identifying a Disjointed Section

In order for a section to be considered a disjointed section, the section has to create multiple solids or multiple cuts. Example of sections which are <u>not</u> disjointed are shown below.

1. Single Cross-Section See Figure 65.
2. Multiple Cross-Sections With One Included in the Other. See Figures 66A and B.

A section which has multiple loops which are not included in an outer loop or are included in multiple outer loops are disjointed. Examples of disjointed sections are shown below.

1. Multiple Outer Loops. See Figure 67A.
2. Multiple Inner Loops. See Figure 67B.

The process for identifying and fixing disjointed sections involves:

1. Find the endpoints of all the entities.
2. Step along entities to form loops.
3. For each loop construct a bounding box.
4. For each loop determine if it is an outer or inner loop.
5. If it is an inner loop, count how many loops it is inside.
6. All outer loops and inner loops contained by an even number of loops will be separated into new sections and new features.

Note: All construction and reference entities are included in each section created for a disjointed section feature.

The following example Is a section which has four loops. Of these loops, two are internal loops. One of the internal loops is completely contained by only one other loop, while the other is contained by two other loops. The two outer loops and the one loop which is contained by two other loops will be separated into new features. Therefore, this section is used to construct three features.

<u>Original Section</u>

See Figure 68.

4 Loops

See Figure 69.

4 Bounding Boxes

See Figure 70.

2 Outer Loops and 2 Inner Loops

See Figure 71.

3 Features

See Figure 72.

TTI NURBS LIBRARY FUNCTIONALITY

Overview

All surface information exported by TTI is represented in NURB Surface Formats. In order to make use of these surfaces for analysis as well as translation data extraction, a NURBS toolkit was developed. This toolkit includes functionality which would be commonly used in utility applications, data extraction and model creation applications written and used by Translation Technologies.

Functionality

The Translation Technologies NURBS toolkit includes the following functionality.

1. General Vector Functions
   - Get the distance between to points.
   - Get the vector between to points.
   - Get the magnitude of a vector.
   - Take the dot product of two vectors.

2. General NURBS Functions
   - Determine the Span on which a certain u or v value is contained.
   - Determine the non zero basis functions governing the surface for a certain u or v value.
   - Evaluate the 3D position (x, y, z) of a NURB surface for a given u and v value.
   - Iteratively solve for the 2D parameter values (u, v) given a 3D point (x, y, z).

- Determine the nearest knot value to a 3D position (x, y, z).
- Count the number of unique knot points on the surface.
- Retrieve a list of unique knot points on the surface.
- Calculate the directional first derivatives (dx/du, dy/du, dz/du) and (dx/dv, dy/dv, dz/dv) of the NURB surface at any point (u,v).
- Calculate the directional second derivatives (d^2x/du^2, d^2y/du^2, d^2z/du^2) and (d^2x/dv^2, d^2y/dv^2, d^2z/dv^2).

3. Other Functionality
   - Import NURBS data from SGF TTI File format.
   - Determine 2D (u,v) bounding box of trimmed NURB surface.

Library Function Definitions (please see header file for argument explanations)

1. double getdistance(double *, double *);
2. void get_vector(double *, double *, double *);
3. double take_magnitude(double *);
4. double dot_product(double *v, double *);
5. int FindSpan(int, double, double *);
6. void BasisFuns(int, double, int, double *, double *, int);
7. void SurfacePoint(struct Nurbs_struct *,double,double,double *);
8. void get_uv_from_x_y_z(struct Nurbs_struct *,double *,double *);
9. void get_nearest_knot(struct Nurbs_struct *, double *, double *);
10. int count_unique_knots(double *, int);
11. void get_unique_knots (double *, double *, int);
12. void get_first_der(struct Nurbs_struct *, double *, double *, double *, double *,double *, double *);
13. void get_second_der(struct Nurbs_struct *, double *, double *,double *,double *, double *);
14. void read_nurbs_data(FILE *, FILE *, struct Nurbs_struct *);
15. void determine_u_v_bound(Trim Info *);

Associated Structures

- Bounding Box Structure

```
typedef struct
{
        double u_min;
        double u_max;
        double v_min;
        double v_max;
}Bounding_Box;
```

- NURBS structure

```
struct Nurbs_struct
```

```
{
        int u_degree;
        int v_degree;
        double *u_knots;
        double *v_knots;
        double u_min;
        double u_max;
        double v_min;
        double v_max;
        int number_of_u_knots;
        int number_of_v_knots;
        int number_of_u_control;
        int number_of_v_control;
        double *control_points;
        Trim_Info trim_info;

        /*Note, weights are included in the control points array.
        ____________________________________________________*/
};
```

- Trim Information

```
typedef struct
{
        int u_patch_id;
        int v_patch_id;
        double min_u_trim;
        double max_u_trim;
        double min_v_trim;
        double max_v_trim;
        int number_of_contours;
        Contour_Info *contour;
}Trim_Info;
```

- Contour Structure

```
typedef struct
{
        int number_of_edges;
        int *number_of_points_on edge;
        Edge_Info *edges;
}Contour_Info;
```

- Edge Information

```
typedef struct
{
        int orientation;
        int number_of_points_on_edge;
        double **tesselation_points;
```

}Edge_Info;

Explanation of Advanced NURBS Toolkit Routines

1. Determining the Span containing a given parameter value.

A NURB surface is generated using two independent parameters, u and v, such that (x, y, z) = f(u, v). Since u and v are independent parameters, determination of the knot span containing a given value of one parameter is not dependent on the other parameter. Therefore, the knot span containing a given parameter value is only dependent on the parameter value and the associated knot sequence of the parameter in question. U and V Spans are independent of each other as shown in Figure 73.

Because the two parameters are independent of each other, the determination of the span on which a given parameter is contained can be treated as a 1D problem. Depending on the degree of the NURB Curve and the desired conditions at the end points there will likely be multiple knots. Translation Technologies in developing the NURBS toolkit does not want to make any assumption as to the end conditions of the NURB. Therefore, the span is calculated using the unique knots. For example, in the figure below there a total of 8 knot values but since there are three knot values at each end of the curve, there are only 3 knot spans. However, in determining the basis functions as described in Part 2 of this section, the number of multiple knots can not be less than the degree of the curve. NURB curve with 8 knots including 3 multiple knots at each end point is shown in Figure 74.

2. Generating the Non-Zero Basis Function

Generating the basis function for a given parameter value is a 1D problem. The basis function associated with a given parameter value is based on the degree of the curve, the knot sequence, the parameter value, and the knot span on which the parameter value is contained.

ASSUMPTION: **It is assumed that the number of knot values at the end points of the curves is greater than or equal to the degree of the curve.**

The knot values involved in generating the basis function will be the first 'n' to either side of the given parameter value where 'n' is the degree of the NURB curve. The figure below shows a NURB curve of degree 3 and the parameter value for which we

want to determine the basis function. The knot values that will be used in generating the basis function will be the 3 on either side of the parameter value. The basis function will be dependent on knots 1,2,3,4,5,6, as shown in Figure 75.

3. Evaluation of NURB Surface To Get a 3D (x, y, z) Point.

Evaluation of a NURB surface to get a 3D point given a 2D point is achieved by multiplying the control points by the basis function for each parameter. Doing this will generate a 4D homogenous point (wx, wy, wz, w). the 3D point is then extracted by dividing wx, wy, and wz by w to get (x, y, z).

4. Obtaining Surface Parameter Values (u,v) Given (x,y,z).

Determining parameter values associated with a given 3D (x,y,z) point involves a root finding method similar to Newton's root finding method. This root finding method is based on the following methodology:

A. Find the nearest knot point and use the associated (u,v) value as an initial guess.

B. Find the derivatives at this position (dx/du, dy/du, dz/du) and (dx/dv, dy/dy, dz/dv).

C. Obtain the vector between the current 3D point and the 3D point of interest.

D. Project this vector onto the two derivative unit vectors.

E. The magnitude of the projected vectors divided by the magnitude of the parameter derivative vectors determines the change in u and v for the next guess.

F. Repeat B-E until the difference between the current 3D point and the 3D point of interest is within tolerance. See Figure 76.

5. Evaluating First Derivative of the NURB Surface at given (u,v)

Evaluation of the First Derivative at a given (u,v) point uses a finite difference approach. The steps involved are fairly simple, and include:

A. Evaluate the (x_0, y_0, z_0) values at the current (u, v) point.
B. Evaluate the (x_1, y_1, z_1) values at the point $(u+\Delta u,v)$.

6. Evaluation of the Second Derivative of the NURB Surface at given (u

,v)

Evaluation of the second derivative at a given (u, v) point is similar to the method for determining the first derivative described above, and includes:

A. Evaluate the First Derivative at the current (u, v) point.

B. Evaluate the First Derivative at the point (u+ Δu, v).

C. Divide the difference in the first derivatives by Δu to get second derivative.

D. Repeat this procedure for the v parameter.

The second derivative calculation is more sensitive to error introduced by having too great a delta value than the first derivative.

File Format Specifications - Supported Features Data Structures

This document defines and explains Applicant's standard file formats and associated data structures used for translation and analysis of CAD models.

1. Introduction and Background

Applicant's CAD conversion process starts with the creation of a set of proprietary neutral files containing the geometry and associated information of a source CAD model. These neutral files are then processed to obtain a native CAD model in the target system. This document identifies the various types of files, the data contained in them and the format of the data. This document also lists the data structures used to store various geometry data during the pre-processing of the source model and during the post processing of the neutral file.

2. Overall Description

The overall translation process is based on the storage of model information in Applicant's file formats. Currently there are seven file formats. The formats and associated purposes are:

1. Model Tree File (.mtf) - Contains information on the model tree or boolean tree.

2. Section Definition File (.sdf) - Contains information on any sketched section associated with features and datums.

3. Surface Geometry File (.sgf) - Contains information on surfaces in the CAD model, both datum surfaces, dumb surfaces, and surfaces associated with solids.

4. Pixie Dust Files (.pxi) - Contain points which lie directly on surfaces and edges.

5. Feature Surface Tree (.fst) - Contains a listing of surfaces in the model and associates the surfaces with the feature they are define (possible only with feature based systems).

6. U Curve files (.Ucrv) - Contain contour lines in the u parameter direction for each surface or face.

V Curve files (.Vcrv) - Contain contour lines in the v parameter direction for each surface or face.

The information contained in the seven file formats completely defines a CAD model. These files are used both for translation and accuracy evaluation of the translated model.

3. Model Tree File (MTF)

The model tree file or MTF file is the standard file format for storing feature/boolean tree information. Currently there are 34 recognized features supported in the MTF file format. These features are:

- CYLINDER,
- CUBOID,
- SPHERE,
- CONE,
- TORUS,
- PYRAMID,
- HOLE,
- ROUND,
- CHAMFER_EDGE,
- CHAMFER_CORNER,
- TWEAK,
- EXTRUSION,
- THIN_EXTRUSION,
- REVOLUTION,
- SWEPT_SOLID,
- VARIABLE_SECTION_SWEEP,
- BLEND,
- NECK,
- FLANGE,
- RIB,
- SHELL,
- DRAFT,
- PIPE,
- DUMB_SOLID,

- DATUM_POINT,
- DATUM_CURVE,
- DATUM_PLANE,
- DATUM_COORD_SYS,
- DATUM_AXIS,
- DATUM_SURFACE,
- DATUM_GENERIC,
- FEATURE_TYPE_UNKNOWN,
- UNSUPPORTED_PROTRUSION,
- UNSUPPORTED_CUT

The MTF file supports features that are either:

- fully defined by a set of parameters,
- sketch based features,
- or surfaces.

In the case of features that are totally defined by a set of parameters, the MTF file contains all the information necessary to build this feature in the target CAD system. Sketch based features (i.e. Extrusions, Revolutions, Sweeps, etc.) will have a reference to a Section Definition File (.sdf) for each sketched portion of the feature.

The MTF file starts with the line "UNITS MM". The next line contains the model name. The body of the file contains information about each individual feature. The last line in the MTF file contains the string "END_FILE". The following is a listing of the some of the supported features, their associated data structures and sample listings in the MTF format.

3.1 Cylinder

3.1.1 Structure

```
#ifndef CYLINDER_STRUCT
#define CYLINDER_STRUCT
        struct Cylinder_struct
                {
                enum Cylinder_Type my_subtype;
                float outer_radius;
                float inner_radius;
                float start_point[3];
```

```
                    float end_point[3];
                    float length;
                    char comment[256];
                    /*The following data members are only used for
                    "my_subtype" = = CYLINDER_COMPLEX_ENDS
                    ____________________________________________*/
                    int start_surface_id;
                    int end_surface_id;
                    };
        #endif
```

3.1.2 MTF Entry

FEATURE	25328			
NUM_SECTIONS	0			
NUM_SURFACES	2			
SURFACE_IDS	25336	25338		
BOOLEAN	SUBTRACTION			
FEATURE_TYPE	CYLINDER			
OUTER_RADIUS	24.000000			
INNER_RADIUS	0.000000			
START_POINT	369.734009	-3.864000	225.035995	
END_POINT	347.544586	-3.864000	187.554550	
LENGTH	43.557178			
COMMENT				

3.2 Cuboid

3.2.1 Structure

```
#ifndef CUBOID_STRUCT
#define CUBOID_STRUCT
    struct Cuboid_struct
        {
        float origin[3];
        float x_axis[3];
        float y_axis[3];
        float z_axis[3];
        float x_length;
        float y_length;
        float z_length;
        char comment[256];
        };
#endif
```

3.2.2 MTF Entry

```
FEATURE  25328
        NUM_SECTIONS        0
        NUM_SURFACES        4
        SURFACE_IDS  25336  25338  14  12  56  7890
        BOOLEAN             UNION
        FEATURE_TYPE        CUBOID
        X_AXIS      1.000000    0.000000    0.000000
        Y_AXIS      0.000000    1.000000    0.000000
        Z_AXIS      0.000000    0.000000    1.000000
        X_ LENGTH43.557178
```

Y_LENGTH 45.9

Z_LENGTH 42.112

COMMENT

3.3 Sphere

3.3.1 Structure

```
#ifndef SPHERE_STRUCT
#define SPHERE_STRUCT
    struct Sphere_struct
        {
        float origin[3];
        float radius;
        float inner_radius;
        char comment[256];
        };
#endif
```

3.3.2 MTF Entry

FEATURE	25328		
	NUM_SECTIONS	0	
	NUM_SURFACES	2	
	SURFACE_IDS	25336	25338
	BOOLEAN	UNION	
	FFATURE_TYPE	SPHERE	
	ORIGIN	1.000000	0.000000 0.000000
	RADIUS	43.557178	
	INNER_RADIUS	0.000000	

COMMENT

3.4 Cone

3.4.1 Structure

```
#ifndef CONE_STRUCT
#define CONE_STRUCT
    struct Cone_struct
        {
        enum Cone_Type my_subtype;
        float base_radius;
        float tip_radius;
        float start_point[3];
        float end point[3];
        float length;
        /*The following data is only used when "my_subtype" = =
        CONE_COMPLEX_ENDS
        ____________________________________________"/
        int start_surface_id;
        int end_surface_id;
        char comment[256];
        };
#endif
```

3.4.2 MTF Entry

FEATURE	25328	
	NUM_SECTIONS	0
	NUM_SURFACES	2
	SURFACE_IDS	25336 25338

BOOLEAN UNION

FEATURE_TYPE CONE

BASE_RADIUS 43.557178

TIP_RADIUS 1.000

START_POINT 1.000000 0.000000 0.000000

END_POINT 100.0000 0.000000 0.000000

LENGTH 99.0

COMMENT

3.5 Torus

3.5.1 Structure

```
#ifndef TORUS_STRUCT
#define TORUS_STRUCT
    struct Torus_struct
        {
        float major_radius;
        float minor_radius;
        float origin[3];
        float start_point[3];
        float end_point[3];
        char comment[256];
        };
#endif
```

3.5.2 MTF Entry

FEATURE 25328

NUM_SECTIONS 0

NUM_SURFACES 2

SURFACE_IDS 25336 25338

BOOLEAN UNION

FEATURE_TYPE TORUS

MAJOR_RADIUS 43.557178

MINOR_ RADIUS 5.0

ORIGIN 0.000000 0.000000 0.000000

START_POINT 1.000000 0.000000 0.000000

END_POINT 100.0000 0.000000 0.000000

COMMENT

3.6 Pyramid

3.6.1 Structure

```
#ifndef PYRAMID_STRUCT
#define PYRAMID_STRUCT
    struct Pyramid_struct
        {
        float vertex[3];
        char comment[256];
        };
#endif
```

3.6.2 MTF Entry

FEATURE 25328

NUM_SECTIONS 1

NUM_SURFACES 5

SURFACE_IDS 25336 25338 4 98 1204

BOOLEAN UNION

FEATURE_TYPE PYRAMID

VERTEX 1.000000 0.000000 0.000000

COMMENT

3.7 Extrusion

3.7.1 Structure

```
#ifndef EXTRUSION_STRUCT
#define EXTRUSION_STRUCT
    struct Extrusion_struct
        {
        enum Extrusion_Type my_subtype;
        char comment[256];
        float start_distance;
        float end_distance;
        /*The following data members are used for my_subtype = = EX_COMPLEX_ENDS
        ____________________________________________*/
        int start_surface_id;
        int end_surface_id;
        };
#endif
```

3.7.2 MTF Entry

FEATURE 58930

NUM_SECTIONS 1

NUM_SURFACES 4

SURFACE_IDS 58935 58945 58947 58951

BOOLEAN UNION

FEATURE_TYPE EXTRUSION

START_DISTANCE -44.662937

END_DISTANCE 0.000000

COMMENT 1 Thru Next - *VERIFY VALUE: 44.662937

3.8 Revolution

3.8.1 Structure

```
#ifndef REVOLUTION_STRUCT
#define REVOLUTION_STRUCT
    struct Revolution_struct
        {
        enum Revolution_Type my_subtype;
        float start_point[3];
        float end_point[3];
        float line_start point[3];
        float line_end point[3];
        int direction;
        char comment[258];

        float start_angle;
        float end_angle;
        /*The following data members are used for my_subtype = =
        REV_COMPLEX_ENDS
        ____________________________________________*/
        int start_surface_id;
```

```
                int end_surface_id;
                };
#endif
```

3.8.2 MTF Entry

```
FEATURE  36350
         NUM_SECTIONS      1
         NUM_SURFACES      5
         SURFACE_IDS       36371  36373  36375  36397  36401
         BOOLEAN           SUBTRACTION
         FEATURE_TYPE      REVOLUTION
         START_POINT       -0.316188   -64.886246   -336.101196
         END_POINT         4.003835    -99.619675   -564.153809
         LINE_START        1.843823    -82.252960   -450.127502
         LINE_END          0.061301    -46.173458   -356.028717
         DIRECTION         -1
         START_ANGLE       0.000000
         END_ANGLE         360.000000
         COMMENT
```

3.9 Swept Solid

3.9.1 Structure

```
#ifndef SWEPT_SOLID_STRUCT
#define SWEPT_SOLID_STRUCT
        struct Swept_Solid_struct
                {
```

```
        int trajectory_id;
        int cross_section_id;
        char comment[256];
        };
    #endif
```

3.9.2 MTF Entry

```
    FEATURE 66082
        NUM_SECTIONS        2
        NUM_SURFACES        0
        SURFACE_IDS
        BOOLEAN             SUBTRACTION
        FEATURE_TYPE        SWEPT_SOLID
        TRAJECTORY_ID 0
        CROSS_SECT_ID       1
        COMMENT
```

3.10 Variable Section Sweep

3.10.1 Structure

```
    #ifndef VARIABLE_SECTION_SWEEP_STRUCT
    #define VARIABLE_SECTION_SWEEP_STRUCT
        struct Variable_Section_Sweep_struct
        {
        int main_trajectory_id;
        int*reference_trajectory_ids;
        int cross_section_id;
        char comment[256];
```

```
};

#endif
```

3.10.2 MTF Entry

```
FEATURE 66062
        NUM_SECTIONS        6
        NUM SURFACES        0
        SURFACE_IDS
        BOOLEAN             SUBTRACTION
        FEATURE_TYPE        VAR_SECTION_SWEEP
        MAIN_TRAJECTORY_ID  0
        CROSS_SECT_ID       1
        REF_TRAJECTORY_IDS2 3 4 5
        COMMENT
```

3.11 Round

3.11.1 Structure

```
#ifndef ROUND_STRUCT
#define ROUND_STRUCT
    struct Round_struct
    {
    int number_of_radii;
    double *radii;
    int number_of_edges;
    double **points;
    char comment[256];
```

```
};

#endif
```

3.11.2 MTF Entry

```
FEATURE 30
        NUM_SECTIONS        0
        NUM_SURFACES        1
        SURFACE_IDS 31
        BOOLEAN             UNION
        FEATURE_TYPE        ROUND
        NUMBER_OF_RADII     1
        RADII 6.000000
        NUMBER_OF_EDGES     1
        POINT       0.000000    0.000000    0.000000
        POINT       1.000000    0.000000    0.000000
        POINT       2.000000    0.000000    0.000000
        POINT       3.000000    0.000000    0.000000
        COMMENT
```

The MTF file is either read into or written out of the structures detailed above. These structures are all part of a linked list of all the features in the model. The structure of the linked list is:

```
#ifndef FEATURE_STRUCT
#define FEATURE_STRUCT
    struct Feature_struct
        {
        int Feature_ID;
```

```
enum Feature_Type my_feature_type;
enum Boolean_Type my_boolean_type;
struct Feature_struct *Next_Feature;
int number_of sections;
int number_of_surfaces;
int *section_ids;
int *my_surface_ids;
struct Section_struct *my_sections;

struct Cylinder_struct *my_cylinder_data;
struct Cuboid_struct *my_cuboid_data;
struct Sphere_struct *my_sphere_data;
struct Cone_struct *my_cone_data;
struct Torus_struct *my_torus_data;
struct Pyramid_struct *my_pyramid_data;
struct Extrusion_struct *my_extrusion_data;
struct Revolution_struct *my_revolution_data;
```

The SDF file starts with the line "SECTION_ID #," where "#" is the section id. The next line contains the section type "SECTION_TYPE 2D," or "SECTION_TYPE 3D." The start of the body of the file contains information about the section plane if the section type is 2D. The next entry in the SDF is the number of entities in the section. The remainder of the body of the SDF contains information about each individual entity. The following is a list of the some entity structures and associated entries in the SDF format. It is important to note that all structures are created in a generic 3D fashion. When these entities are used in 2D cross sections, the z value should be zero as each entity is sketched in the local coordinate system of the sketching plane.

4.3 LINE

4.3.1 Structure

```
#ifndef LINE_STRUCT
#define LINE_STRUCT
    struct Line_struct
        {
        enum Line_Type my_line_type;
        float start_point[3];
        float end point[3];
        };
#endif
```

4.3.2 SDF Entry

```
ENTITY_ID       15
    ENTITY_TYPE LINE
        LINE_TYPE   CENTERLINE
        START_POINT     448.720001  -33.864498  0.000000
        END_POINT       448.720001  84.374496   0.000000
```

Or for a non-centerline:

```
ENTITY_ID       2
    ENTITY_TYPE LINE
        LINE_TYPE GEOMETRY
        START_POINT     405.383209  51.676739   0.000000
        END_POINT       410.572266  28.594715   0.000000
```

4.4 ARC

4.4.1 Structure

```
#ifndef ARC_STRUCT
#define ARC_STRUCT
    struct Arc_struct
        {
        float origin[3];
        float normal[3];
        float v1[3];
        float v2[3];
        float radius;
        float start angle;
        float end_angle;
        };
#endif
```

4.4.2 SDF Entry

```
ENTITY_ID 6
    ENTITY_TYPE ARC
        ORIGIN          371.235474  44.000000   0.000000
        NORMAL          0.000000    0.000000    1.000000
        VECTOR_1 1.000000   0.000000    0.000000
        VECTOR_2 0.000000   1.000000    0.000000
        RADIUS          35.000000
        START_ANGLE     12.670000
        END_ANGLE       90.000000
```

The start and end angles are always measured with respect to v1 in the direction of the normal.

4.5 ELLIPTICAL_ARC

4.5.1 Structure

```
#ifndef ELLIPTICAL_ARC_STRUCT
#define ELLIPTICAL_ARC_STRUCT
    struct Elliptical_Arc_struct
        {
        float origin[3];
        float normal[3];
        float v1[3];
        float v2[3];
        float major_radius;
        float minor_radius;
        float twist_angle;
        float start_angle;
        float end_angle;
        };
#endif
```

4.5.2 SDF Entry

ENTITY_ID 6

ENTITY_TYPE ELLIPTICAL_ARC

ORIGIN	371.235474	44.000000	0.000000
NORMAL	0.000000	0.000000	1.000000

VECTOR_1 1.000000 0.000000 0.000000

VECTOR_2 0.000000 1.000000 0.000000

MAJOR_RADIUS 35.000000

MINOR_RADIUS 15.000000

TWIST_ANGLE 5.000000

START_ANGLE 90.000000
END_ANGLE 270.000000

4.6 CIRCLE

4.6.1 Structure

```
#ifndef CIRCLE_STRUCT
#define CIRCLE_STRUCT
    struct Circle_struct
        {
        float center[3];
        float radius;
        float origin[3];
        float normal[3];
        float v1[3];
        float v2[3];
        };
#endif
```

4.6.2 SDF Entry

ENTITY_ID 6

ENTITY_TYPE CIRCLE

CENTER -51.719101 -371.922333 0.000000

RADIUS 6.000000

ORIGIN 0.000000 0.000000 0.000000

NORMAL 0.000000 0.000000 1.000000

VECTOR_1 1.000000 0.000000 0.000000
VECTOR 2 0.000000 1.000000 0.000000

4.7 SPLINE

4.7.1 Structure

```
#ifndef SPLINE_STRUCT
#define SPLINE_STRUCT
    struct Spline_struct
        {
        float start_angle;
        float end_angle;
        int number_of_points;
        struct Point_struct *my_points;
        float **tangents;
        };
#endif
```

4.7.2 SDF Entry

ENTITY_ID 23

ENTITY_TYPE SPLINE

START_ANGLE 175.259613

END ANGLE 2.569920

NUM_POINTS 7

POINT 20.463680 443.485352 0.000000

POINT	19.126945	443.591431	0.000000
POINT	17.789448	443.688019	0.000000
POINT	16.451263	443.775146	0.000000
POINT	15.112475	443.852783	0.000000
POINT	13.773165	443.920929	0.000000
POINT	12.433411	443.979584	0.000000
TANGENT	-24.053049	1.994588	0.000000
TANGENT	-24.067551	1.824114	0.000000
TANGENT	-24.080599	1.653542	0.000000
TANGENT	-24.092197	1.482869	0.000000
TANGENT	24.102345	1.312098	0.000000
TANGENT	-24.111038	1.141226	0.000000
TANGENT	-24.118280	0.970256	0.000000

Note: Start and End angles are no longer used for spline definition.

4.8 POINT

4.8.1 Structure

```
#ifndef POINT_STRUCT
#define POINT_STRUCT
    struct Point_struct
        {
        float location[3];
        };
#endif
```

4.8.2 SDF Entry

ENTITY_ID 3

ENTITY_TYPE POINT

LOCATION 0.000000 1.000000 0.000000

4.9 COORD_SYS

4.9.1 Structure

```
#ifndef COORD_SYS_STRUCT
#define COORD_SYS_STRUCT
    struct Coord_sys_struct
        {
        float location[3];
        float x_axis[3];
        float y_axis[3];
        float z_axis[3];
        };
#endif
```

4.9.2 SDF Entry

ENTITY_ID 6

ENTITY_TYPE COORD_SYS

LOCATION 371.235474 44.000000 0.000000

X_AXIS 1.000000 0.000000 0.000000

Y_AXIS 0.000000 1.000000 0.000000

Z_AXIS 0.000000 0.000000 1.000000

4.10 PLANE

4.10.1 Structure

```
#ifndef PLANE_STRUCT
#define PLANE_STRUCT
    struct Plane_struct
        {
        float v1[3];
        float v2[3];
        float normal[3];
        float origin[3];
        };
#endif
```

4.10.2 SDF Entry

```
ENTITY_ID       2
ENTITY_TYPE PLANE
        VECTOR_1  0.997966    0.000000    -0.063751
        VECTOR_2  0.062081    -0.227387    -0.971824
        NORMAL          -0.014496    0.973804    -0.226925
        ORIGIN          -329.667633  130.255264  432.975983
```

Only a Section_Plane will exist in a 2D section. However, a 3D section may have a plane as an entity.

The section structure has a list of all entities in the section. The list is in the form of an Entity Union structure.

```
#ifndef ENTITY_STRUCT
#define ENTITY_STRUCT
    union Entity_struct
```

```
{
struct Plane_struct my_plane;
struct Line_struct my_lines;
struct Arc_struct my_arcs;
struct Elliptical_Arc_struct my_elliptical_arcs;
struct Circle_struct my_circles;
struct Spline_struct my_splines;
struct Point_struct my_points;
struct Coord_sys_struct my_coord_sys;
};
#endif
```

The entire Section Structure is:

```
#ifndef SECTION_STRUCT_H
#define SECTION_STRUCT_H
struct Section_struct
{;
int number_of_entities;
int my_section_id;
int my_feature_id;
enum Section_Type my_section_type;
enum Entity_Type *my_entity_types;
int *entity_ids;

/*Entity Structures for storing the data.
_____*/
struct Plane_struct my_plane;
```

```
union Entity_struct *my_entities;
};
```

#endif

5. Feature Surface Tree

The Feature Surface Tree file (FST) is a listing of all the surface ids associated with a model. Surfaces are created for the entire model, not just the features which are surfaces. Whenever possible the FST breaks the list of surfaces up into lists of ids for each surface. If, however, the source CAD system is not a feature based system, the surfaces will be associated with each solid. A sample FST from a feature based system looks like the figure below.

```
                    Source-motorout
                    FEATURE 7
Source model        NURBS 10
                    NURBS 13
                    NURBS 16
                    NURBS 18
                    ENDFEATURE 7
  Surface Ids       FEATURE 22
                    NURBS 30                Feature IDs
                    NURBS 32
                    ENDFEATURE 22
                    FEATURE 42
                    NURBS 51
                    NURBS 53
                    ENDFEATURE 42
                    ENDFILE
```

6. The same model from a non feature based CAD system would look like the next figure.

```
                    Source-motorout
                    FEATURE 1
                    NURBS 10
                    NURBS 13                Feature ID is
                    NURBS 16                now a solid ID
                    NURBS 18
  Surface Ids       NURBS 30
                    NURBS 32
```

NURBS 51
NURBS 53
ENDFEATURE 1

7. Surface Geometry File

The Surface Geometry File (SGF) contains the information on each individual surface. The file contains two types of information, surface information and trim information. The only surface type currently supported in the SGF format is a NURB surface. A NURB surface is able to exactly replicate any surface. The SGF file format is diagramed below.

The first section in the SGF contains header information about the generation of the individual surface. The next set of information involves the control points for the NURB surface. The control point coordinates are in homogenous 4D, coordinates (wx,wy,wz,w).

Header Information	#NURBS #ORIGIN PROE #1 = Entity Number #53 = Entity id #42 = Feature id
Orientation of the surface is corrected before exporting from the CAD system	###Orientation of surface = -1 ################## #Control point array ##################
Number of control points in each paramente direction	4 = # of u control points 4 = # of v control points
Control Points (wx, wy, wz, w)	0.500000 -4.000126 -12.000000 2.000000 = Control pt 0 0.500000 -1.333375 -12.000000 2.000000 = Control pt 1 0.500000 1.333375 -12.000000 2.000000 = Control Pt 2 0.500000 4.000126 -12.000000 2.000000 = Control pt 3 0.166667 -1.333375 -4.333333 0.666667 = Control Pt 4 0.166667 -0.444458 -4.333333 0.666667 = Control Pt S 0.166667 0.444458 -4.333333 0.666667 = Control pt 6 0.166667 1.333375 -4.333333 0.666667 = Control pt 7 -0.166667 -1.333375 -4.333333 0.666667 = Control Pt 8 -0.166667 -0.444458 -4.333333 0.666667 = Control pt 9 -0.166667 0.444458 -4.333333 0.666667 = Control pt 10 -0.166667 1.333375 -4.333333 0.666667 = Control Pt 11 -0.500000 -4.000126 -12.000000 2.000000 = Control pt 12 -0.500000 -1.333375 -12.000000 2.000000 = Control pt 13 -0.500000 1.333375 -12.000000 2.000000 = Control Pt 14 -0.500000 4.000126 -12.000000 2.000000 = Control Pt 15

The control points are generated keeping u constant and incrementing v until v = v_max. Then v is reset to v_min and u is incremented.

The next set of surface information contains the knot vectors for u and v and the degree of the surface in each direction.

	##########################
	#Knot array
	##########################
Knot vector dimensions	8 = # of u knots
	8 = # of v knots
Knot vector for u parameter	0.000000 = Knot Pt 0
	0.000000 = Knot Pt 1
	0.000000 = Knot Pt 2
	0.000000 = Knot Pt 3
	1.000000 = Knot Pt 4
	1.000000 = Knot Pt 5
	1.000000 = Knot Pt 6
	1.000000 = Knot Pt 7
Knot vector for v parameter	-4.40063 = Knot Pt 0
	-4.40063 = Knot Pt 1
	-4.40063 = Knot Pt 2
	-4.40063 = Knot Pt 3
	-0.399937 = Knot Pt 4
	-0.399937 = Knot Pt 5
	-0.399937 = Knot Pt 6
	-0.399937 = Knot Pt 7
Degree of the surface	3 = Degree in U
	3 = Degree in V

This is the end of the surface information. At this point all information necessary to build the surface has been recorded in the SGF. Trim information is started in the next section of the SGF. Trim information enables CAD systems and Applicant's analyzers to determine which parts of the surface defined in the first part of the SGF is used in association with other surfaces to create a solid, or trimmed face in the source CAD system.

The first set of trim information records the number of contours that trim the surface. There can be only one outer contour, but the number of internal contours (holes) is not limited.

Separator between	##########################

surface information and trim information
#Trim Information
####################

Number of trim contours 2 = # of Contours

Next, the SGF contains information about each contour. The trim contour information is broken up into edges. The SGF records the number of edges per trim contour and then the edge information for each edge. The edge information is written to the SGF in two formats. The first method is B-spline trim curves for each edge. At the present time, this information is not used due to problems getting the B-spline information to be in the u-v space of the NURB surface. The second form of trim information is currently the one being used by Applicant. This trim information is written as points in the u-v space of the NURB that lie on the trim edge. Later, a trim curve can be fit through these u-v points to form a good approximation of the trimming edge.

Contour Separator

Contour 0

Number of Edges
= # of edges

Edge Separator
Edge 0
****

B-SPLINE

Number of B-spline control points
2 = # of u control points

Control points
-0.25000 1.98431 -6.00000 1.0
-0.25000 1.98431 -6.02026 1.0

Number of knots
4 = # of u knots

Knot Vector
0.00000
0.00000
1.00000
1.00000

Number of points
2 = # uv values on surface corresponding to edge

#U, V, and X Y Z of the tessellated curve

u, v and x,y,z point	0.000000	0.000000	-0.25000	1.98431	-6.00000
on trimming edge	0.000000	1.000000	-0.25000	1.98431	-6.00000

This structure will be repeated for each trim contour and each edge inside of each contour.

CLAIMS

1. A computational geometry system, comprising:

a server having processing circuitry and including an operation manager configured to compare source geometric data in a source geometric model with target geometric data in a target geometric model, and operative to identify discrepancies in the geometric data therebetween;

a communication link;

at least one client communicating with the server over the communication link; and

an interrupt interface provided by one of the at least one client and the server and operative to notify a user of the presence of an inability to automatically generate an accurate representation of the source geometric model in the target geometric model.

2. The computational geometry system of claim 1 wherein the interrupt interface notifies a user of the presence of any identified discrepancies in response to comparing the geometric data.

3. The computational geometry system of claim 1 wherein the operation manager comprises a medium having program code embodied therein which when executed by the processing circuitry translates a source geometric model received from a source within a client/server environment to a target geometric model.

4. The computational geometry system of claim 3 wherein the source geometric data is derived from the source geometric model and the target geometric data is derived from the target geometric model.

5. The computational geometry system of claim 3 wherein the medium comprises memory coupled to the processing circuitry to store the source geometric model and the target geometric model in computer files.

6. The computational geometry system of claim 3 wherein the program code is associated with the processing circuitry and the interrupt interface, the program code including a production control module operative to implement staged translation of the source geometric model into the target geometric model.

7. The computational geometry system of claim 6 wherein the staged translation comprises: program code for extracting comparison reference data from the source geometric model in a source CAD system; program code for generating a target geometric model in a target CAD system; program code for comparing reference data from the source geometric model with corresponding reference data from the target geometric model; and upon identification of a discrepancy, program code for displaying the discrepancy to an operator at the client with the interrupt interface.

8. The computational geometry system of claim 7 wherein the program code for comparing reference data comprises program code for extracting point cloud data from the source geometric model and comparing the extracted point cloud data with geometry in the target geometric model.

9. The computational geometry system of claim 7 wherein the program code for comparing reference data comprises program code for extracting point cloud data from the target geometric model and comparing the extracted point cloud data with geometry in the source geometric model.

10. The computational geometry system of claim 7 wherein the program code for generating a target geometric model comprises program code for generating a user interrupt at the interrupt interface responsive to identifying a problem in generating the target geometric model.

11. A computational geometry system, comprising:

a client/server environment;

a client having an interrupt interface;

a server communicating with the client via the environment and having processing circuitry and an operation manager configured to compare source geometric

data in a source geometric model with target geometric data in a target geometric model, and operative to identify discrepancies in the geometric data between the source geometric model and the target geometric model;

wherein the interrupt interface is operative to notify a user of the presence of an identified discrepancy in response to comparing the geometric data or encountering a problem in creating the target geometric model.

12. The computational geometry system of claim 11 wherein the operation manager further comprises program code for performing actions, including: evaluating architecture of the source geometric model including decomposing a model of the source geometric model.

13. The computational geometry system of claim 12 further comprising program code for examining construction history detailing the manner in which the source geometric model was built.

14. The computational geometry system of claim 13 further comprising program code for extracting the source geometric data from the source geometric model.

15. The computational geometry system of claim 11 further comprising program code for generating the target geometric data based upon a construction history used to create the source geometric model.

16. The computational geometry system of claim 11 further comprising program code for generating an interrupt at the interrupt interface to alert a user of the client of the presence of an identified discrepancy in the geometric data between the source geometric model and the target geometric model or a problem in creating the target geometric model.

17. An interrupt interface, comprising:

a server having a database configured to store input data of a source model, processing circuitry configured to convert the input data of the source model

into corresponding output data of a target model, and a source and target model comparator configured to compare the input data with the corresponding output data and identify geometric discrepancies between the input data and the output data;

at least one client having a user interface and communicating with the server; and

a production controller having common interface production control software configured to serially interrupt a user via the user interface at one of the clients when at least one geometric discrepancy or creation problem has been identified.

18. The interrupt interface of claim 17 wherein the source and target model comparator comprises a model comparator.

19. The interrupt interface of claim 18 wherein the model comparator comprises program code for performing actions, including implementing a forward comparison check by extracting point cloud data from the source model and comparing the extracted point cloud data with the corresponding target model.

20. The interrupt interface of claim 18 wherein the model comparator comprises program code for performing actions, including implementing a backward comparison check by extracting point cloud data from the target model and comparing the extracted point cloud data with the corresponding source model.

21. The interrupt interface of claim 17 wherein, after identifying discrepancies between the input data and the output data or a creation problem, the production controller is operative to save the output data of the target model in a memory location and identify geometric discrepancies present within the output data of the target model.

22. The interrupt interface of claim 21 wherein a user accesses the saved output data of the target model from the memory location via the user interface of the client in order to inspect the saved output data.

23. A method for creating a target geometric model from a source geometric model, comprising:

providing a server and a client of a computational geometry system having a user interface;

extracting source geometric data from the source geometric model file;

storing the extracted source geometric data in a metafile format;

using a target CAD system, generating a target geometric model having target geometric data;

detecting at least one of a) a discrepancy between the models and b) a problem in generating the target geometric model while generating the target geometric model; and

generating an interrupt at the user interface responsive to detecting the at least one of a discrepancy between the models and a problem in generating the target geometric model.

24. The method of claim 23 wherein, after generating an interrupt, interrupting generation of the target geometric model.

25. The method of claim 24 wherein, after generating an interrupt, further comprising clearing the interrupt via the user interface.

26. The method of claim 25 wherein, after clearing the interrupt, continuing to generate the target geometric model.

27. The method of claim 23 wherein generating an interrupt comprises stopping generation of the target geometric model and displaying a notice to a user at the user interface requesting assistance with one of the discrepancy and the problem.

28. The method of claim 27 wherein generating the target geometric model comprises substantially duplicating a process used to create the source geometric model based at least in part on identified architecture, mathematical basis, and definition of the geometry of the source geometric model.

29. The method of claim 23 wherein, prior to extracting source geometric data, receiving a pre-existing source geometric model at the server and storing the source geometric model in memory of the server.

30. The method of claim 29 wherein extracting source geometric data comprises evaluating the pre-existing source geometric model to determine architecture and construction history.

31. The method of claim 23 wherein the user interface comprises an interrupt interface of a user display on the client.

32. The method of claim 23 wherein generating an interrupt comprises providing a visual cue within the target CAD system to remove the discrepancy and help fix the geometry.

33. The method of claim 23 wherein the server and the client are provided by a common device.

34. A method for managing computational geometry system translations, comprising:

providing a server and at least one client within a client/server network environment;

receiving source geometric data at the server within a memory;

generating target geometric data using the source geometric data; and

identifying discrepancies between the target geometric data and the source geometric data by comparing the target geometric data with the source geometric data.

35. The method of claim 34 wherein, after identifying a discrepancy, further comprising notifying a user of the client of the discrepancy.

36. The method of claim 34 wherein the source geometric data comprises comparison reference data extracted from a pre-existing source model and the target

geometric data comprises corresponding comparison reference data created in a target model generated using the pre-existing source model.

37. The method of claim 36 wherein the comparison reference data comprises point cloud data.

38. The method of claim 36 wherein, after receiving the pre-existing source model, further comprising opening the source model using a source CAD system.

39. The method of claim 36 wherein the comparison reference data is extracted from the pre-existing source model.

40. The method of claim 38 further comprising storing the extracted comparison reference data in a metafile format.

41. A machine-executed method for implementing a geometric conversion on a computer system including an interface, the method comprising:

receiving a source geometric model at a computer;

storing the source geometric model in memory of the computer;

converting the source geometric model to a target geometric model;

extracting source comparison reference data from the source geometric model;

extracting target comparison reference data from the target geometric model; and

comparing the comparison reference data from one of the source geometric model and the target geometric model with geometry from one of the target geometric model and the source geometric model, respectively, to identify geometric discrepancies therebetween.

42. The method of claim 41 wherein the comparison reference data comprises point cloud data.

43. The method of claim 42 wherein comparing comprises determining whether point cloud data from the target geometric model lies outside of one of a surface, an edge, and a curve of point cloud data from the source geometric model using a predetermined geometric tolerance.

44. The method of claim 42 wherein the point cloud data identifies one of a surface and an edge of a surface.

45. The method of claim 41, responsive to comparing, interrupting a user at the interface when a geometric discrepancy is identified.

46. The method of claim 41, responsive to converting, interrupting a user at the interface when a problem is encountered during converting the source geometric model to the target geometric model.

47. A geometric model comparator, comprising:

processing circuitry configured to generate a target model from a source model;

memory configured to store the source model and the target model; and

comparison circuitry configured to identify selected points from the source model, create corresponding selected points in a target model, and compare the selected points from the source model with the target model to identify geometric entities from the target model that fall outside of a predetermined tolerance range with the respective one or more points from the source model.

48. The comparator of claim 47 wherein the comparison circuitry implements a forward comparison between the selected points from the source model and the respective geometry of the target model.

49. The comparator of claim 47 wherein the comparison circuitry implements a reverse comparison between the geometry of the source model and the respective selected points from the target model.

50. The comparator of claim 47 wherein the comparison circuitry implements a bi-directional comparison between one of the geometry of the source model and the respective selected points from the target model with one of the geometry of the target model and the geometry of the source model, respectively.

51. The comparator of claim 47 wherein the comparison circuitry implements a bounding box comparison comprising a minimized box that fits around a curve comprising the selected points from the target model and the source model so as to enclose the curve.

52. The comparator of claim 47 wherein the comparison circuitry is configured to measure a distance to a nearest edge from a selected point in the target model and the source model, compare the distance in the target model and the source model, and if a difference between the compared distances falls outside of a pre-determined tolerance, initiate notification of a user of the geometric model comparator of a discrepancy between the target model and the source model.

53. The comparator of claim 47 wherein the comparison circuitry saves selected points between the source model and the target model that do not match into an error file comprising a bad edge pixie file.

54. A geometric translation system, comprising:

memory to receive input data of a source model;

processing circuitry configured to convert the input data of the source model into corresponding output data of a target model;

a geometric model comparator configured to compare the input data with the corresponding output data and identify geometric discrepancies between the input data and the output data.

55. The system of claim 54 wherein the processing circuitry extracts comparison reference data from the source model.

56. The system of claim 55 wherein the source model comprises point cloud data.

57. The system of claim 55 wherein the comparison reference data is stored in a metafile format.

58. The system of claim 54 wherein the geometric model comparator is implemented in the processing circuitry.

59. The system of claim 54 wherein the target geometric output file comprises a target CAD file including data points representing the geometric location of at least one of edges and surfaces of the source model.

60. The system of claim 54 wherein the source model comprises pre-existing geometric file data and the target model comprises target geometric file data, and wherein the mirror model comparator compares the pre-existing geometric file data with the target geometric file data.

61. The system of claim 54 further comprising a client and a server provided by a common device.

62. A machine-executed method for implementing a geometric conversion on a computer system including an interface, the method comprising:

receiving a source geometric model at a computer;

storing the source geometric model in memory of the computer;

converting the source geometric model to a target geometric model;

extracting reference data from one of the source geometric model and the target geometric model;

importing reference data into one of the target geometric model and the source geometric model;

comparing the extracted reference data for discrepancies with geometry from one of the target geometric model and the source geometric model in one of a target CAD system and a source CAD system.

63. The method of claim 62 wherein reference data is extracted from the source geometric model, reference data is imported into the target geometric model, and the extracted reference data is compared for discrepancies with geometry from the target geometric model in a target CAD system.

64. The method of claim 62 wherein reference data is extracted from the target geometric model, reference data is imported into the source geometric model, and the extracted reference data is compared for discrepancies with geometry from the source geometric model in a source CAD system.

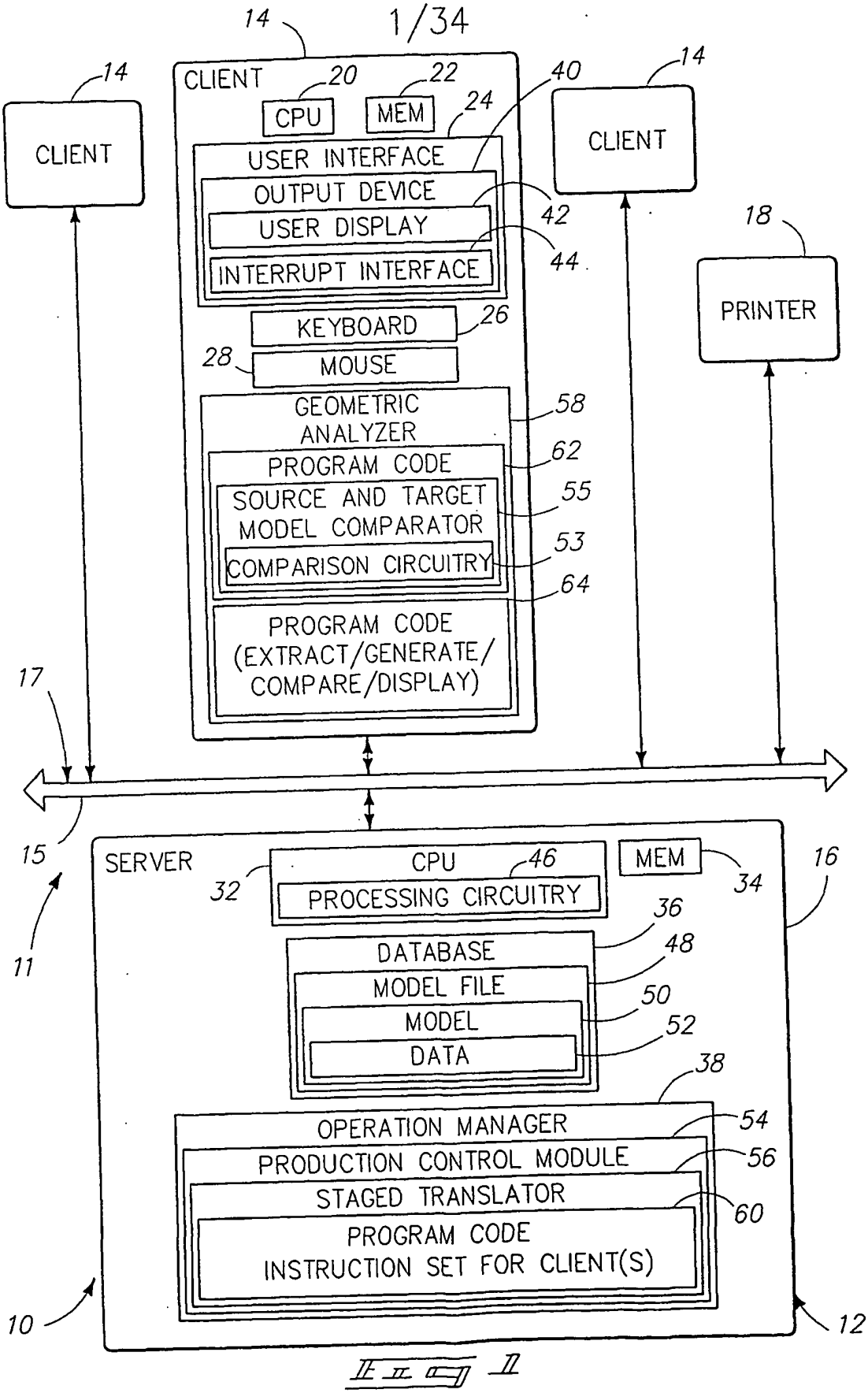

14
CLIENT
14
CLIENT
CLIENT
20
CPU
22
MEM
24
USER INTERFACE
OUTPUT DEVICE
USER DISPLAY
INTERRUPT INTERFACE
40
42
44
KEYBOARD
26
28
MOUSE
GEOMETRIC ANALYZER
58
PROGRAM CODE
62
SOURCE AND TARGET MODEL COMPARATOR
55
COMPARISON CIRCUITRY
53
64
PROGRAM CODE (EXTRACT/GENERATE/ COMPARE/DISPLAY)
18
PRINTER
17
15
11
SERVER
32
CPU
46
PROCESSING CIRCUITRY
MEM
34
16
36
DATABASE
48
MODEL FILE
50
MODEL
52
DATA
38
OPERATION MANAGER
54
PRODUCTION CONTROL MODULE
56
STAGED TRANSLATOR
60
PROGRAM CODE INSTRUCTION SET FOR CLIENT(S)
10
12

Fig. 1

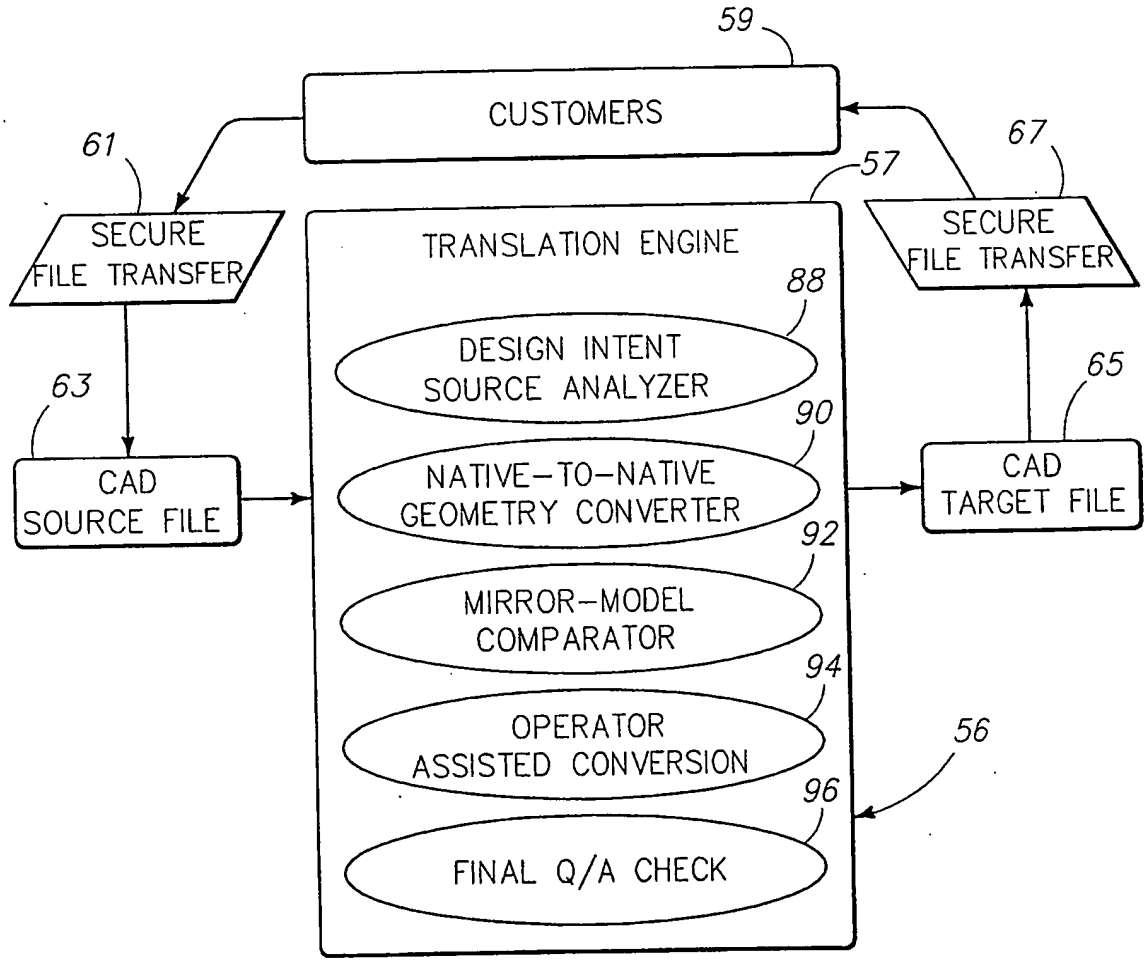

59
CUSTOMERS
61
SECURE FILE TRANSFER
57
TRANSLATION ENGINE
67
SECURE FILE TRANSFER
88
DESIGN INTENT SOURCE ANALYZER
65
63
90
NATIVE-TO-NATIVE GEOMETRY CONVERTER
CAD SOURCE FILE
CAD TARGET FILE
92
MIRROR-MODEL COMPARATOR
94
OPERATOR ASSISTED CONVERSION
56
96
FINAL Q/A CHECK

Fig 2

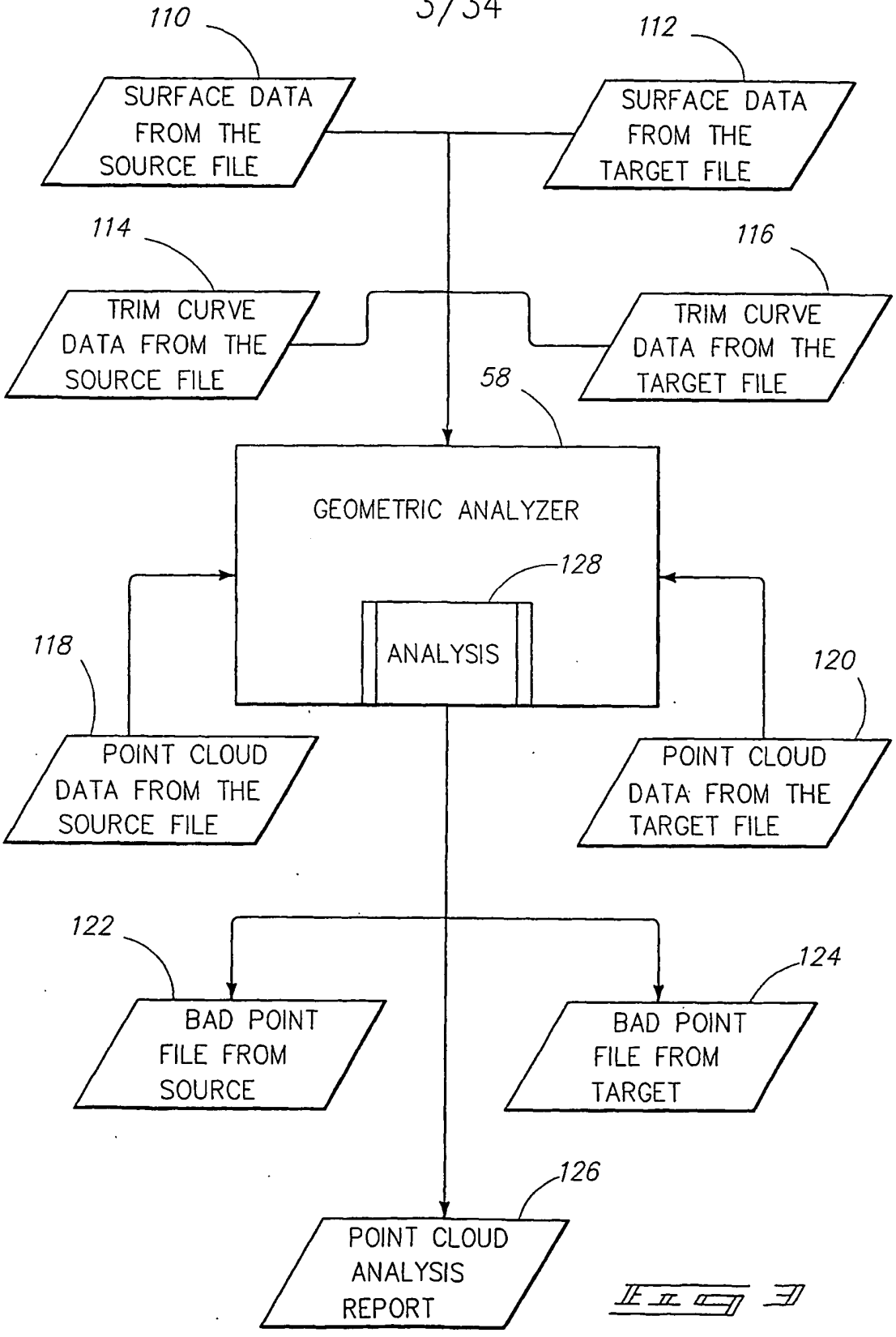

110
SURFACE DATA FROM THE SOURCE FILE
112
SURFACE DATA FROM THE TARGET FILE
114
TRIM CURVE DATA FROM THE SOURCE FILE
116
TRIM CURVE DATA FROM THE TARGET FILE
58
GEOMETRIC ANALYZER
128
ANALYSIS
118
POINT CLOUD DATA FROM THE SOURCE FILE
120
POINT CLOUD DATA FROM THE TARGET FILE
122
BAD POINT FILE FROM SOURCE
124
BAD POINT FILE FROM TARGET
126
POINT CLOUD ANALYSIS REPORT

Fig 3

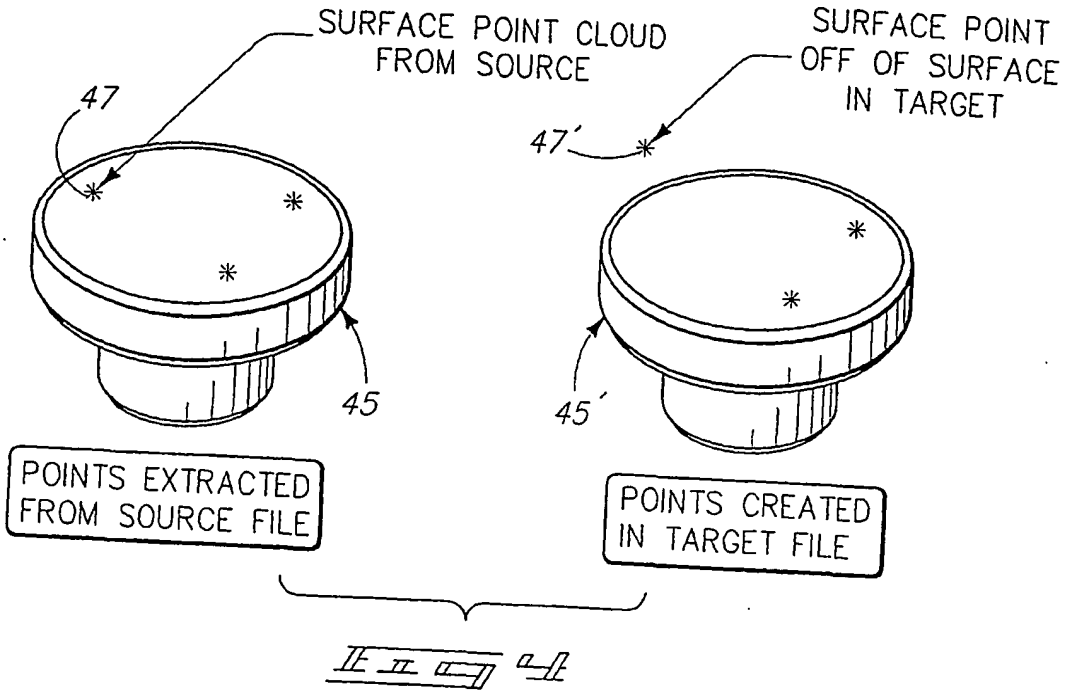

SURFACE POINT CLOUD FROM SOURCE
SURFACE POINT OFF OF SURFACE IN TARGET
47
47'
45
45'
POINTS EXTRACTED FROM SOURCE FILE
POINTS CREATED IN TARGET FILE

Fig 4

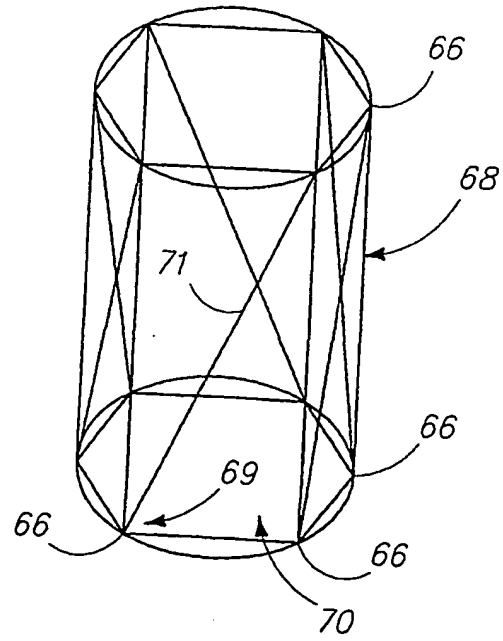

66
68
71
66
69
66
66
70

Fig 5

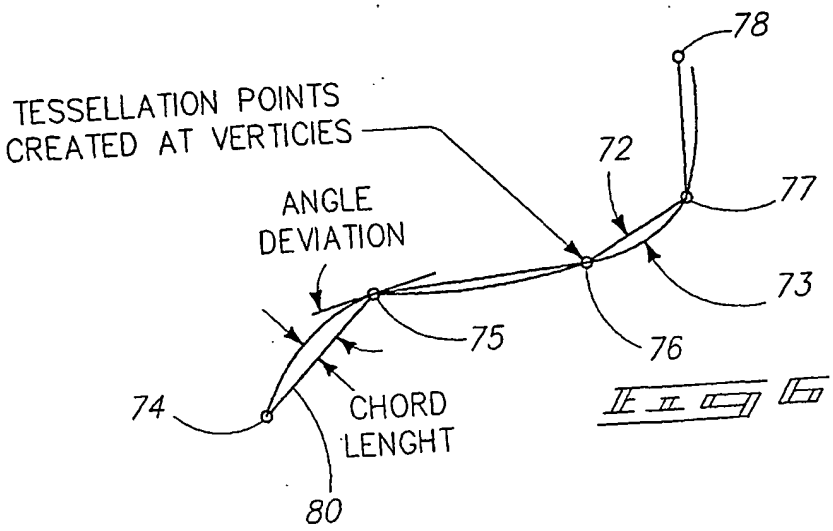

78
TESSELLATION POINTS
CREATED AT VERTICIES
72
77
ANGLE
DEVIATION
73
75
76
74
CHORD
LENGHT
80

Fig 6

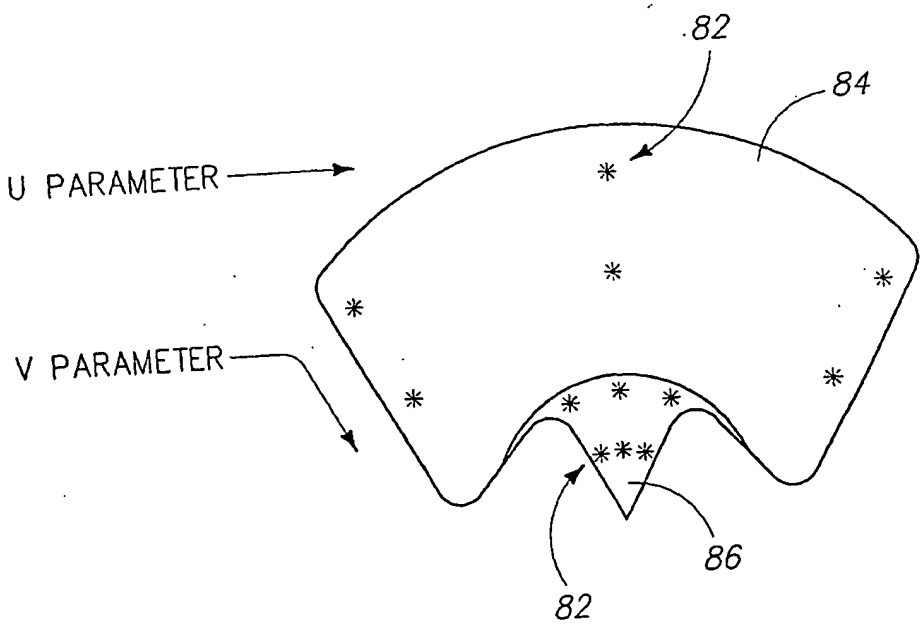

82
84
U PARAMETER
V PARAMETER
86
82

Fig 7

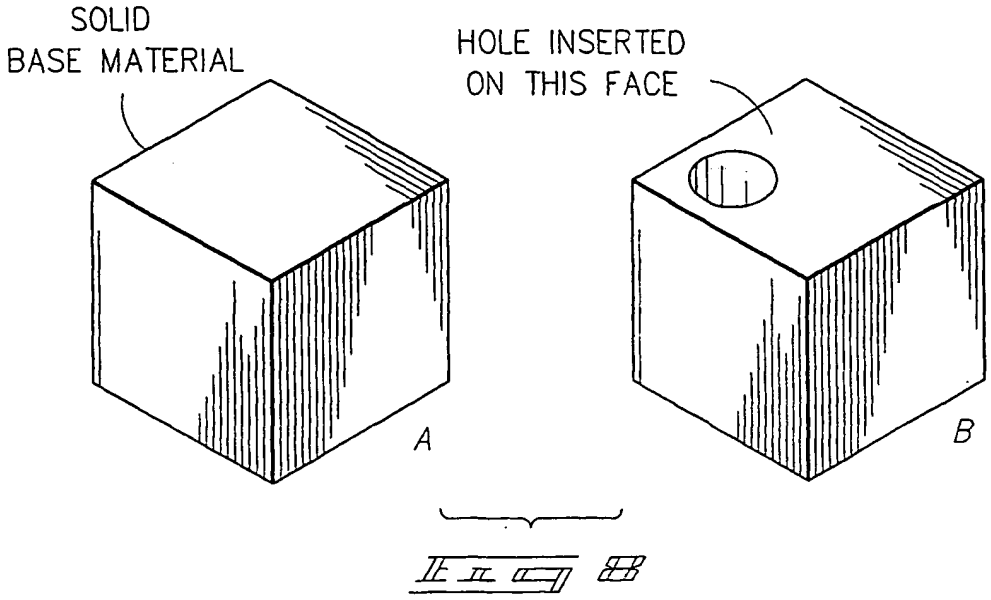

SOLID
BASE MATERIAL
HOLE INSERTED
ON THIS FACE
A
B

Fig 8

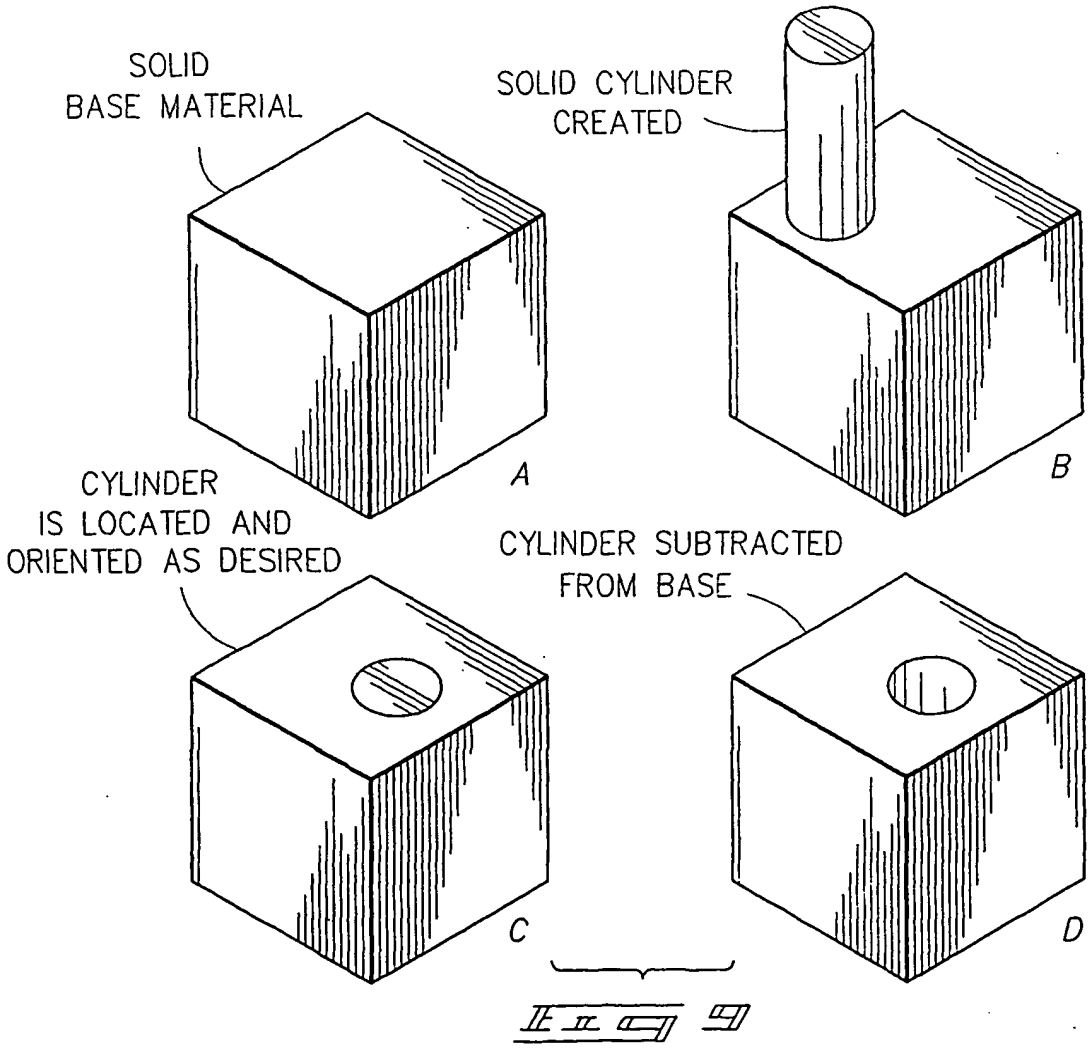

SOLID
BASE MATERIAL
SOLID CYLINDER
CREATED
A
B
CYLINDER
IS LOCATED AND
ORIENTED AS DESIRED
CYLINDER SUBTRACTED
FROM BASE
C
D

Fig 9

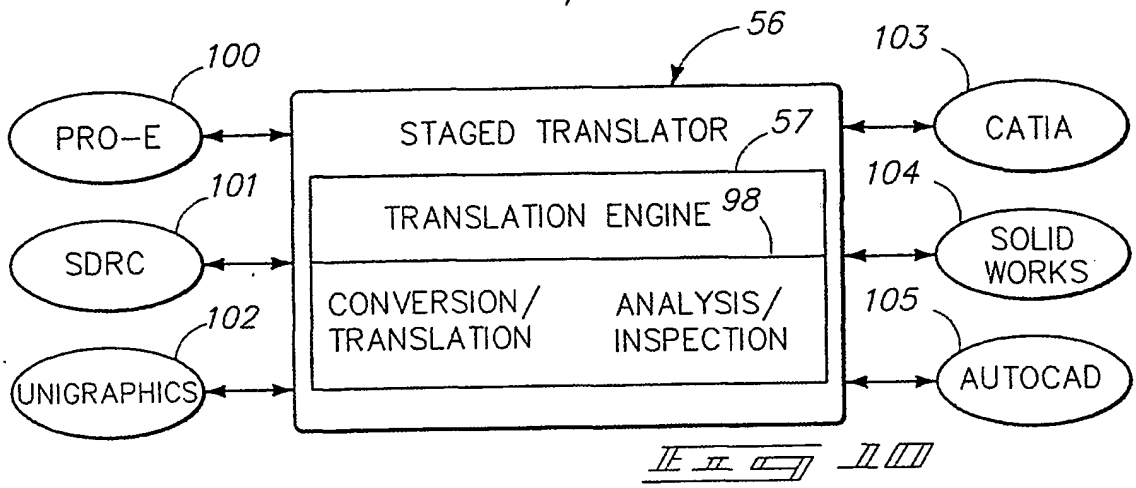


Fig 10

FEATURE BASED TREE

P_1
P_2
P_3
P_4
P_5
C_1
P_6
P_7
P_8
C_2

Fig 11

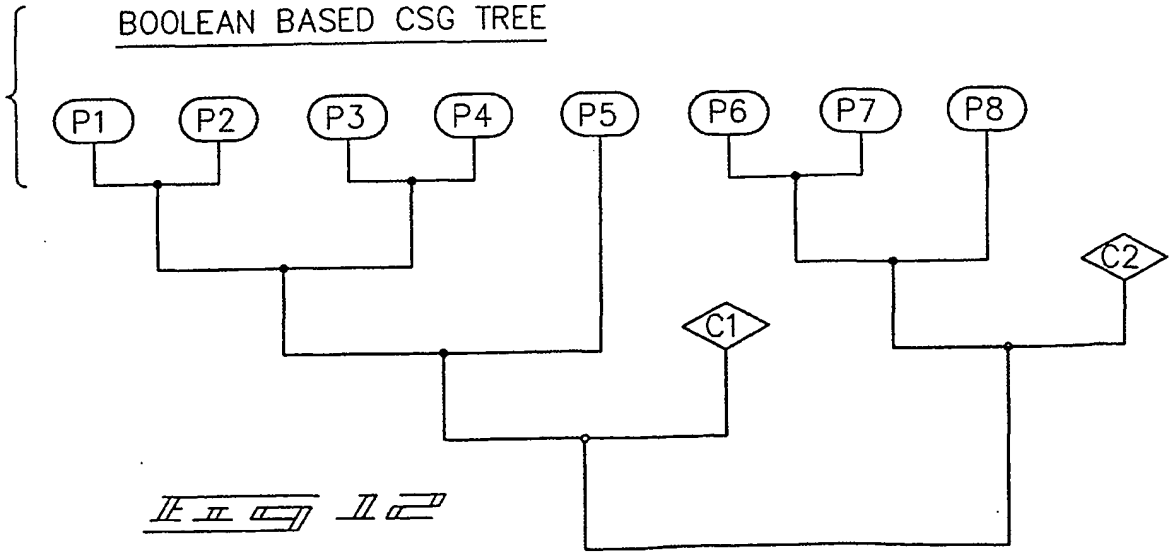


Fig 12

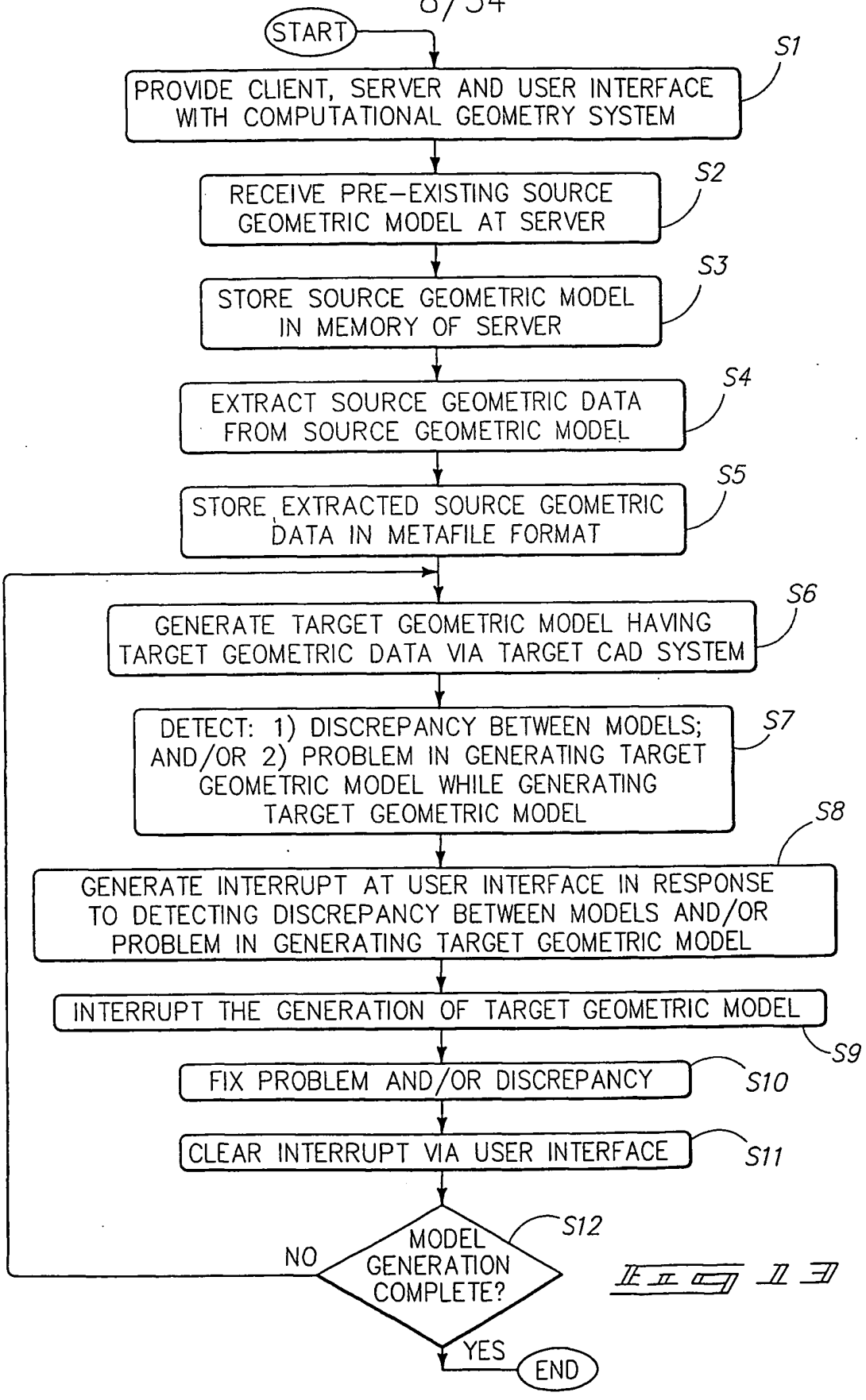

START
S1
PROVIDE CLIENT, SERVER AND USER INTERFACE WITH COMPUTATIONAL GEOMETRY SYSTEM
S2
RECEIVE PRE-EXISTING SOURCE GEOMETRIC MODEL AT SERVER
S3
STORE SOURCE GEOMETRIC MODEL IN MEMORY OF SERVER
S4
EXTRACT SOURCE GEOMETRIC DATA FROM SOURCE GEOMETRIC MODEL
S5
STORE EXTRACTED SOURCE GEOMETRIC DATA IN METAFILE FORMAT
S6
GENERATE TARGET GEOMETRIC MODEL HAVING TARGET GEOMETRIC DATA VIA TARGET CAD SYSTEM
S7
DETECT: 1) DISCREPANCY BETWEEN MODELS; AND/OR 2) PROBLEM IN GENERATING TARGET GEOMETRIC MODEL WHILE GENERATING TARGET GEOMETRIC MODEL
S8
GENERATE INTERRUPT AT USER INTERFACE IN RESPONSE TO DETECTING DISCREPANCY BETWEEN MODELS AND/OR PROBLEM IN GENERATING TARGET GEOMETRIC MODEL
INTERRUPT THE GENERATION OF TARGET GEOMETRIC MODEL
S9
FIX PROBLEM AND/OR DISCREPANCY
S10
CLEAR INTERRUPT VIA USER INTERFACE
S11
S12
MODEL GENERATION COMPLETE?
NO
YES
END

Fig 13

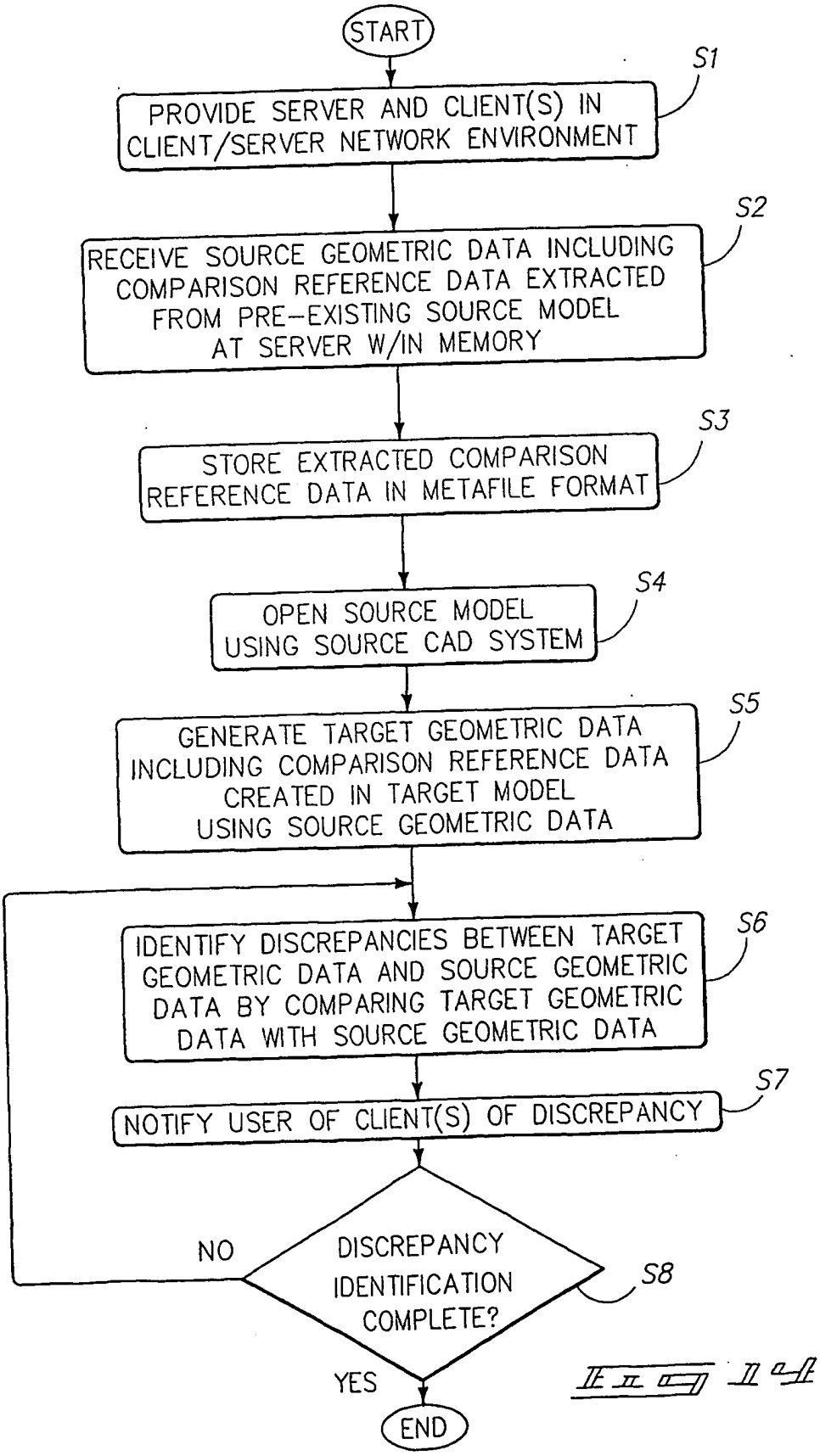

START
PROVIDE SERVER AND CLIENT(S) IN CLIENT/SERVER NETWORK ENVIRONMENT
S1
RECEIVE SOURCE GEOMETRIC DATA INCLUDING COMPARISON REFERENCE DATA EXTRACTED FROM PRE-EXISTING SOURCE MODEL AT SERVER W/IN MEMORY
S2
STORE EXTRACTED COMPARISON REFERENCE DATA IN METAFILE FORMAT
S3
OPEN SOURCE MODEL USING SOURCE CAD SYSTEM
S4
GENERATE TARGET GEOMETRIC DATA INCLUDING COMPARISON REFERENCE DATA CREATED IN TARGET MODEL USING SOURCE GEOMETRIC DATA
S5
IDENTIFY DISCREPANCIES BETWEEN TARGET GEOMETRIC DATA AND SOURCE GEOMETRIC DATA BY COMPARING TARGET GEOMETRIC DATA WITH SOURCE GEOMETRIC DATA
S6
NOTIFY USER OF CLIENT(S) OF DISCREPANCY
S7
DISCREPANCY IDENTIFICATION COMPLETE?
S8
NO
YES
END

Fig. 11A

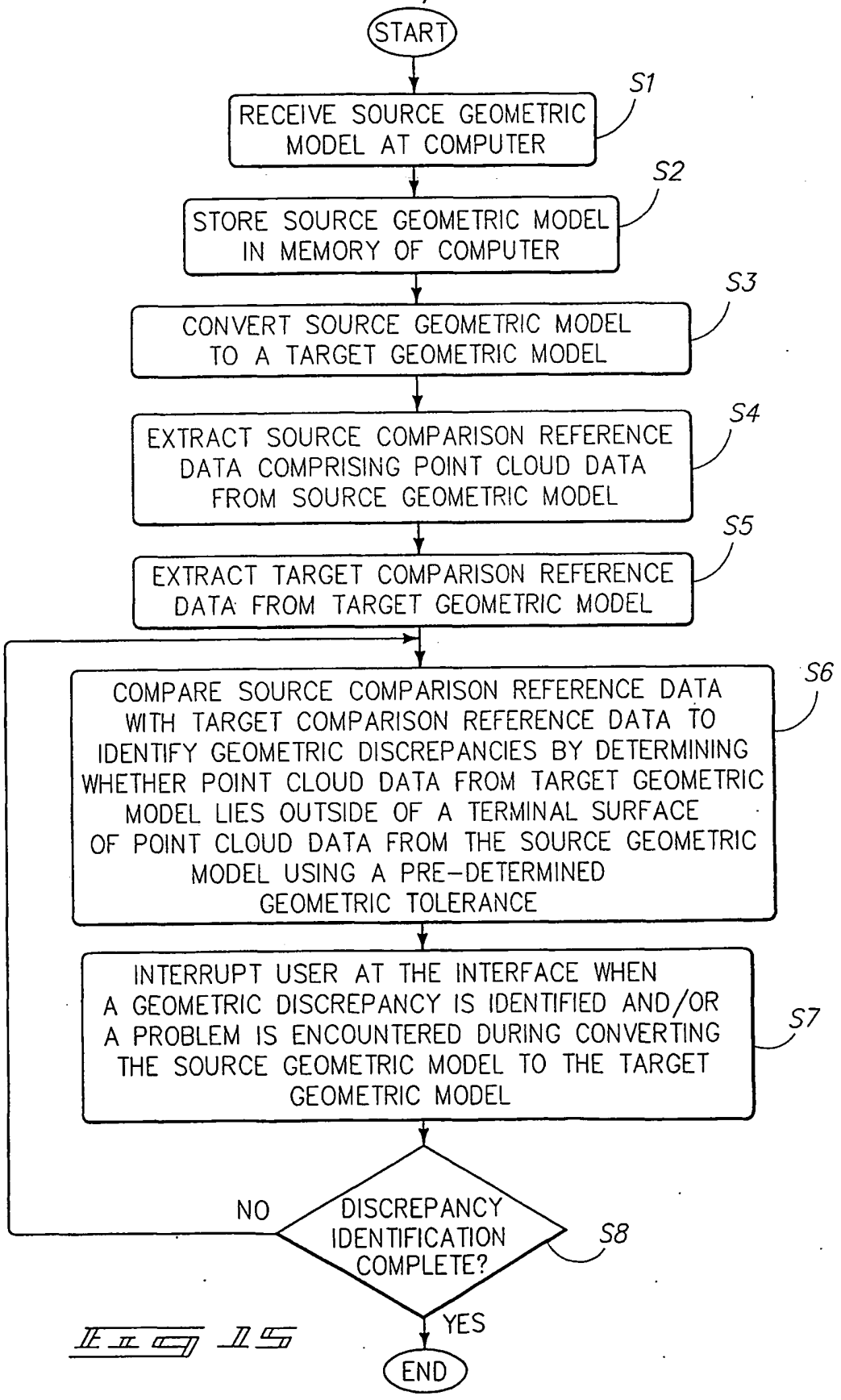

START
RECEIVE SOURCE GEOMETRIC MODEL AT COMPUTER
S1
STORE SOURCE GEOMETRIC MODEL IN MEMORY OF COMPUTER
S2
CONVERT SOURCE GEOMETRIC MODEL TO A TARGET GEOMETRIC MODEL
S3
EXTRACT SOURCE COMPARISON REFERENCE DATA COMPRISING POINT CLOUD DATA FROM SOURCE GEOMETRIC MODEL
S4
EXTRACT TARGET COMPARISON REFERENCE DATA FROM TARGET GEOMETRIC MODEL
S5
COMPARE SOURCE COMPARISON REFERENCE DATA WITH TARGET COMPARISON REFERENCE DATA TO IDENTIFY GEOMETRIC DISCREPANCIES BY DETERMINING WHETHER POINT CLOUD DATA FROM TARGET GEOMETRIC MODEL LIES OUTSIDE OF A TERMINAL SURFACE OF POINT CLOUD DATA FROM THE SOURCE GEOMETRIC MODEL USING A PRE-DETERMINED GEOMETRIC TOLERANCE
S6
INTERRUPT USER AT THE INTERFACE WHEN A GEOMETRIC DISCREPANCY IS IDENTIFIED AND/OR A PROBLEM IS ENCOUNTERED DURING CONVERTING THE SOURCE GEOMETRIC MODEL TO THE TARGET GEOMETRIC MODEL
S7
DISCREPANCY IDENTIFICATION COMPLETE?
S8
NO
YES
END

Fig 15

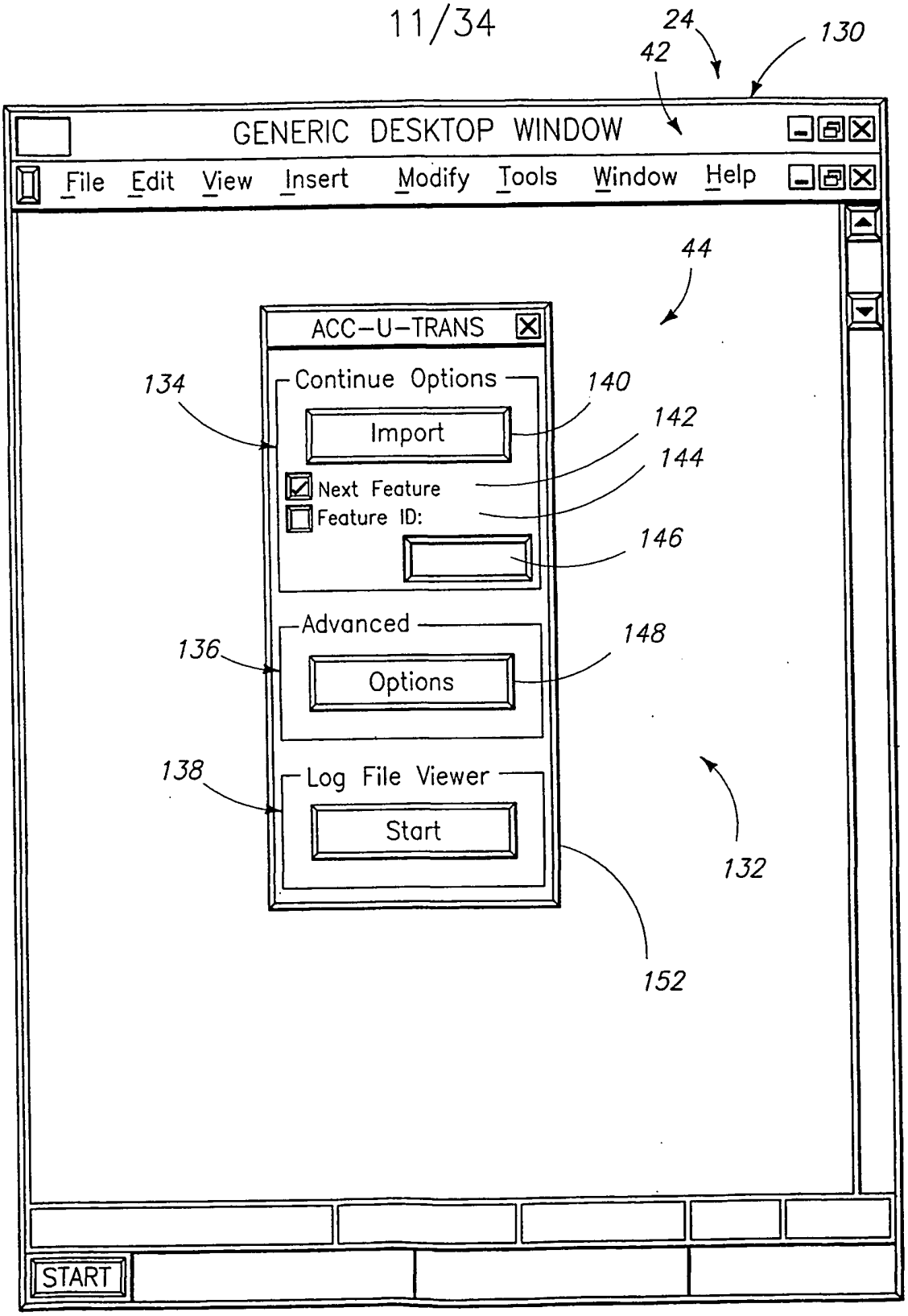

24
130
42
GENERIC DESKTOP WINDOW
File Edit View Insert Modify Tools Window Help
44
ACC-U-TRANS
134
Continue Options
140
Import
142
144
Next Feature
Feature ID:
146
Advanced
136
148
Options
138
Log File Viewer
Start
132
152
START

Fig 16

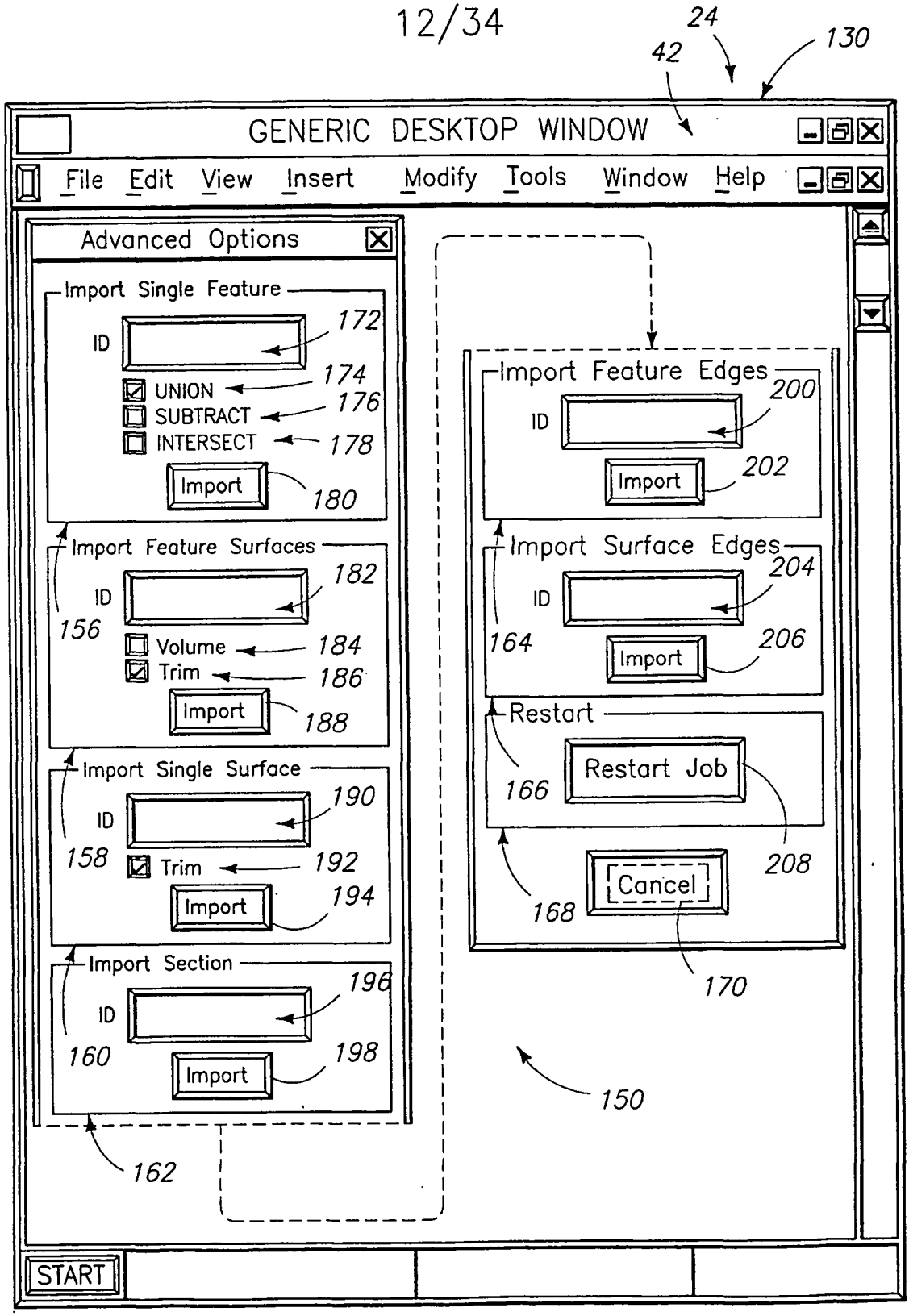

24
42
130
GENERIC DESKTOP WINDOW
File Edit View Insert Modify Tools Window Help
Advanced Options
Import Single Feature
ID
172
UNION
174
SUBTRACT
176
INTERSECT
178
Import
180
Import Feature Surfaces
ID
182
156
Volume
184
Trim
186
Import
188
Import Single Surface
ID
190
158
Trim
192
Import
194
Import Section
ID
196
160
Import
198
162
Import Feature Edges
ID
200
Import
202
Import Surface Edges
ID
204
164
Import
206
Restart
166
Restart Job
208
Cancel
168
170
150
START

Fig 11

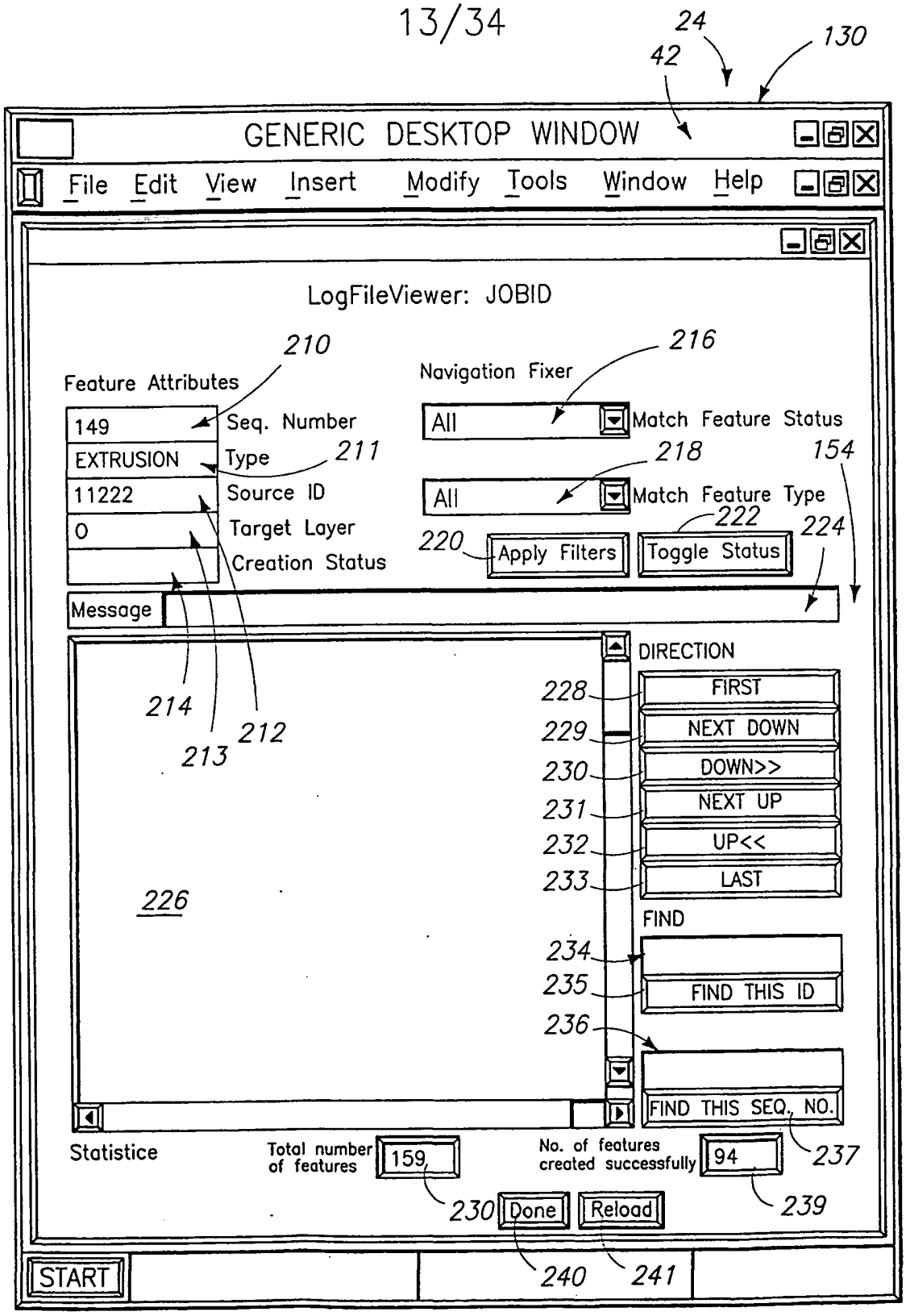

24
130
42
GENERIC DESKTOP WINDOW
File Edit View Insert Modify Tools Window Help
LogFileViewer: JOBID
210
Feature Attributes
149
Seq. Number
EXTRUSION
Type
211
11222
Source ID
0
Target Layer
Creation Status
216
Navigation Fixer
All
Match Feature Status
218
All
Match Feature Type
154
220
Apply Filters
222
Toggle Status
224
Message
214
213
212
DIRECTION
228
FIRST
229
NEXT DOWN
230
DOWN>>
231
NEXT UP
232
UP<<
233
LAST
226
FIND
234
235
FIND THIS ID
236
FIND THIS SEQ. NO.
237
Statistice
Total number of features
159
No. of features created successfully
94
239
230
Done
Reload
240
241
START

Fig 11B

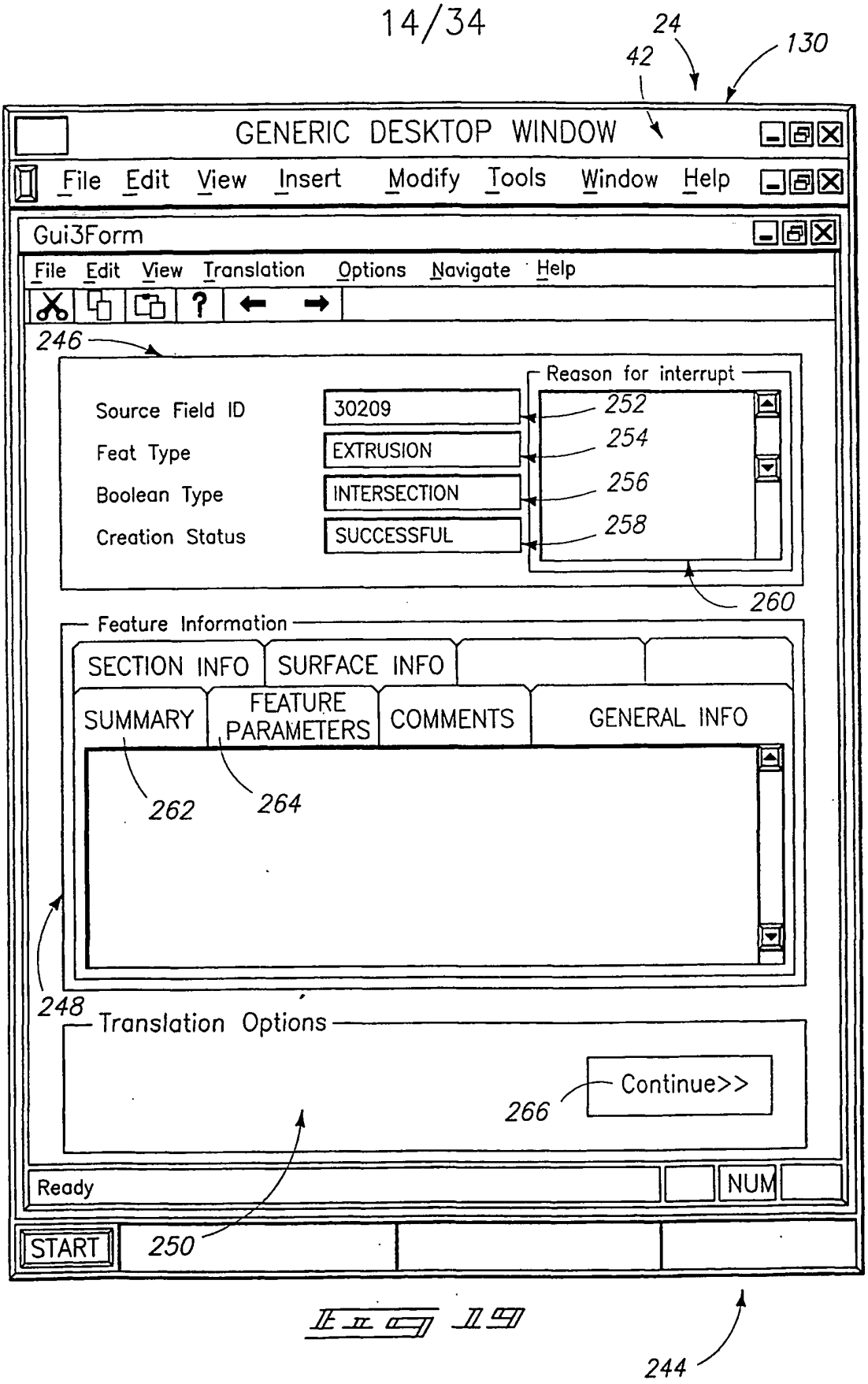

24
42
130
GENERIC DESKTOP WINDOW
File Edit View Insert Modify Tools Window Help
Gui3Form
File Edit View Translation Options Navigate Help
246
Reason for interrupt
Source Field ID
30209
252
Feat Type
EXTRUSION
254
Boolean Type
INTERSECTION
256
Creation Status
SUCCESSFUL
258
260
Feature Information
SECTION INFO
SURFACE INFO
SUMMARY
FEATURE PARAMETERS
COMMENTS
GENERAL INFO
262
264
248
Translation Options
Continue>>
266
Ready
NUM
START
250
244

Fig 19

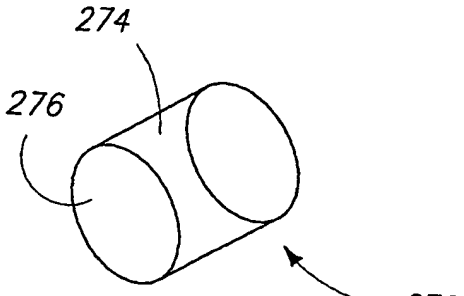


Fig 20

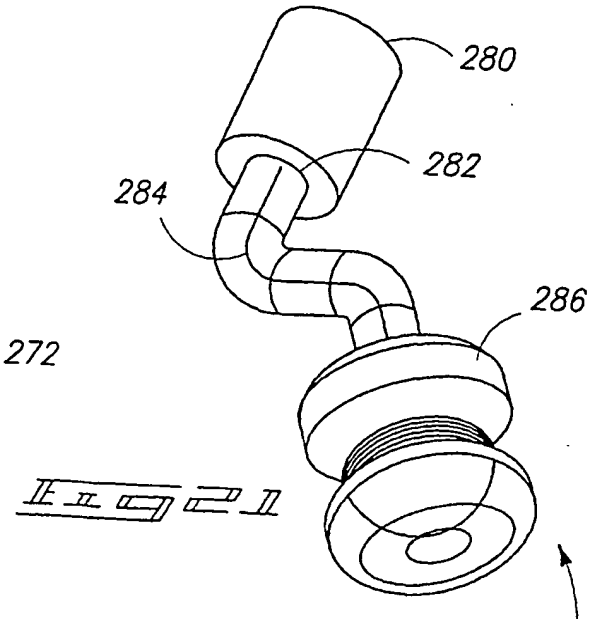


Fig 21

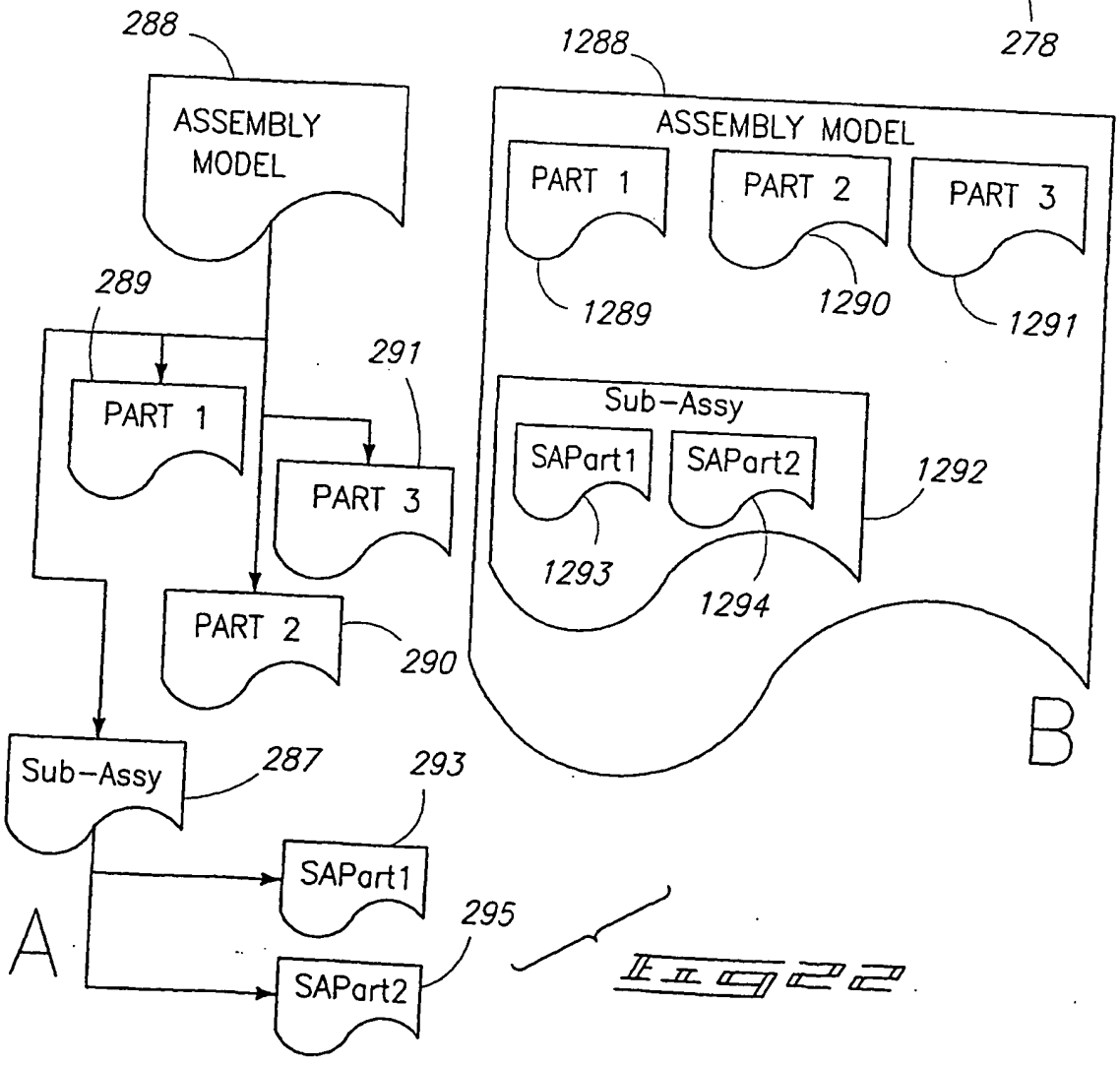


Fig 22

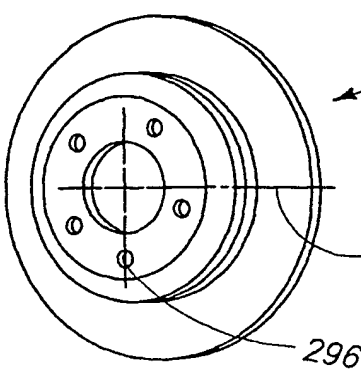


Fig 23

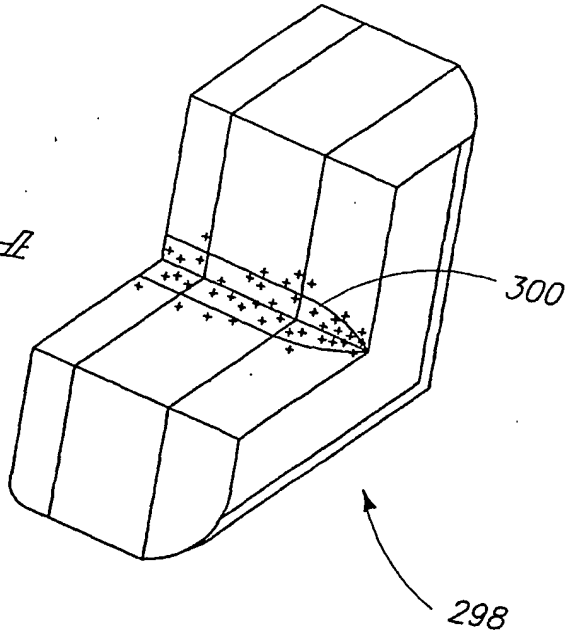


Fig 24

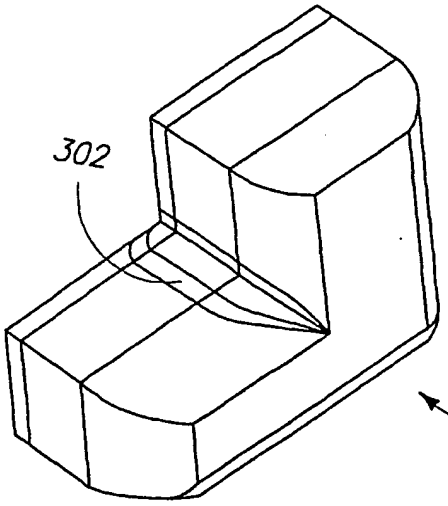


Fig 25

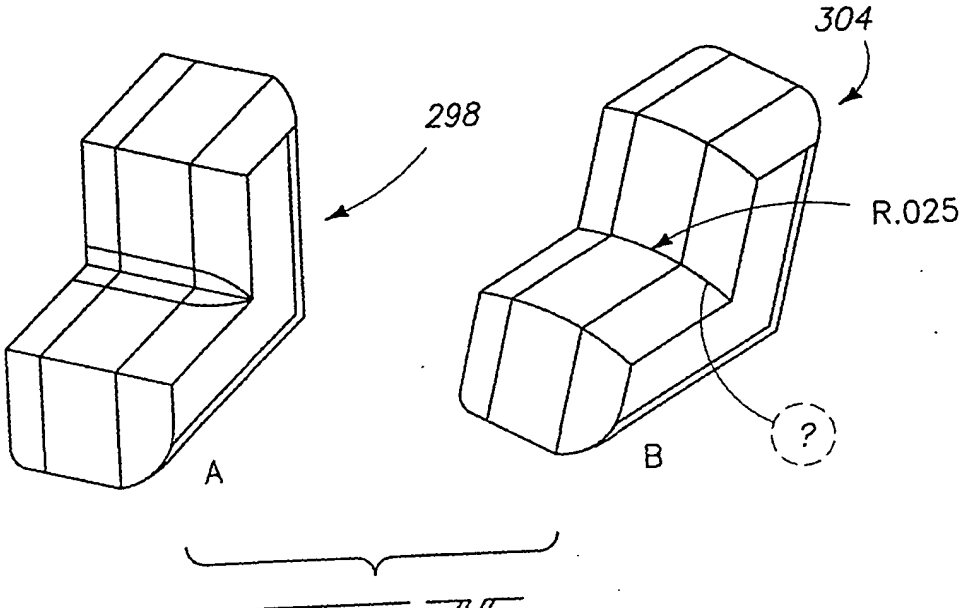


Fig 26

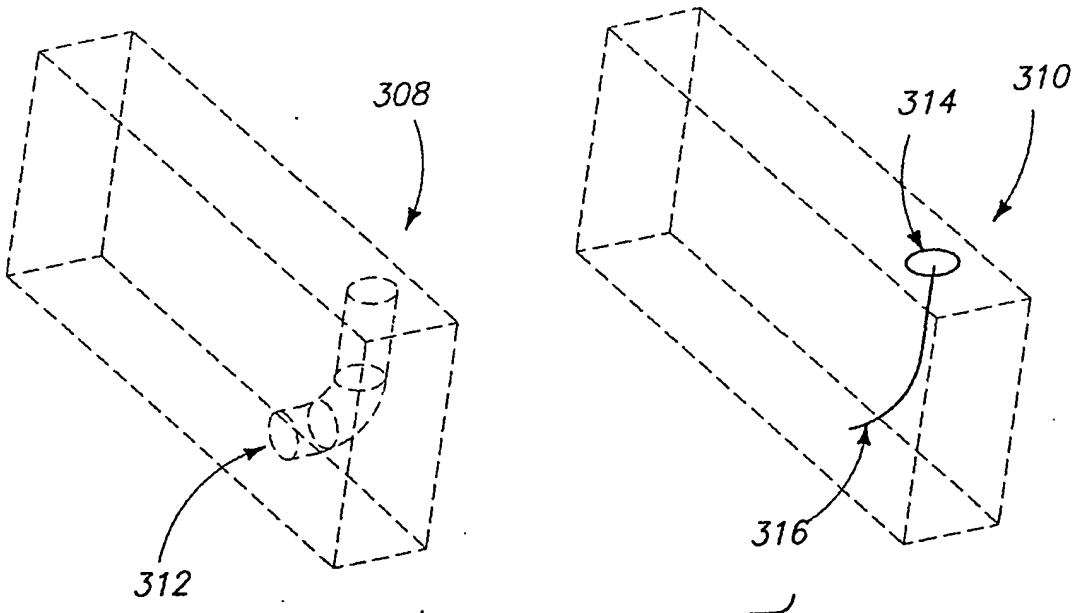


Fig 27

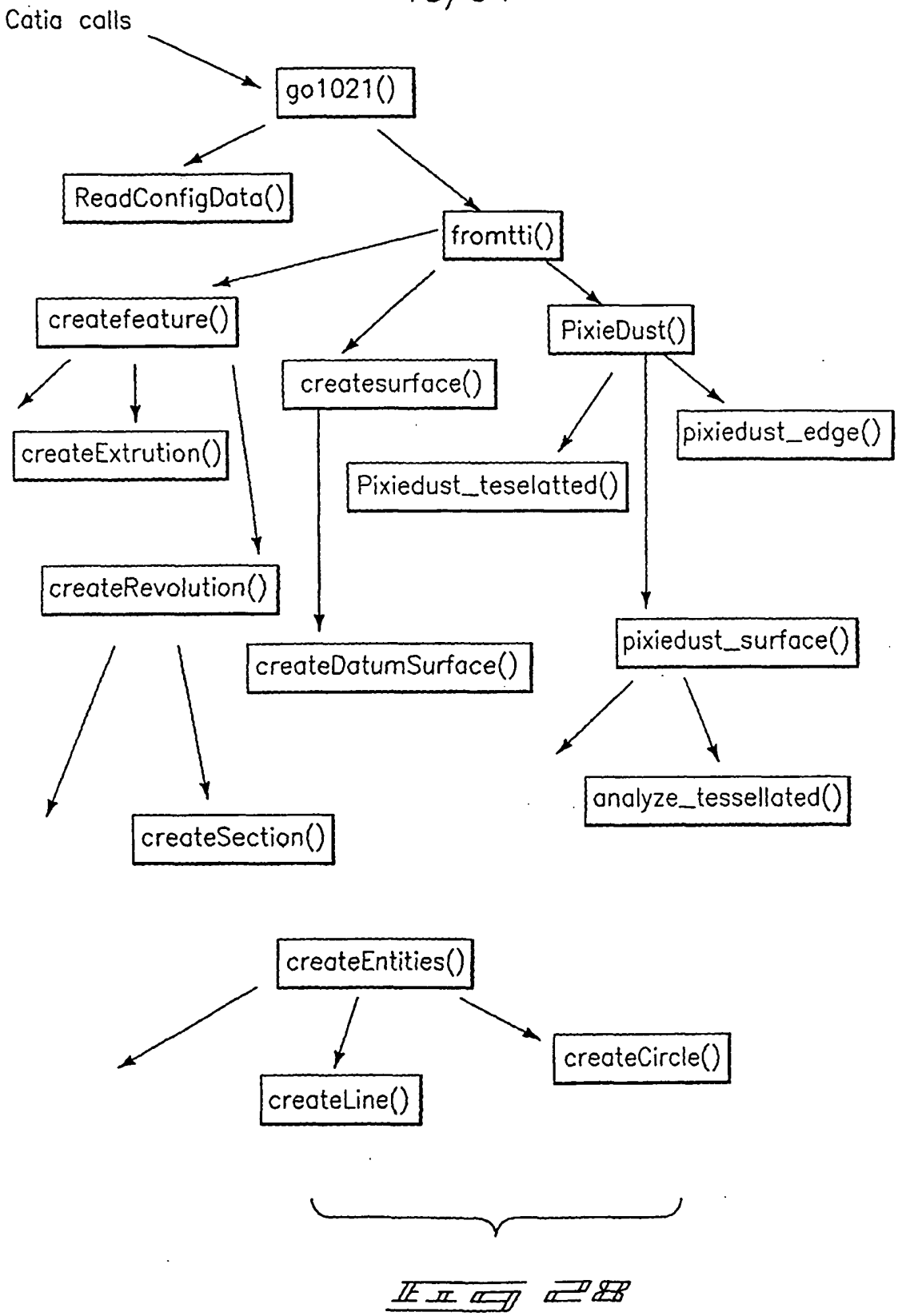

Catia calls
go1021()
ReadConfigData()
fromtti()
createfeature()
createsurface()
PixieDust()
createExtrution()
pixiedust_edge()
Pixiedust_teselatted()
createRevolution()
createDatumSurface()
pixiedust_surface()
analyze_tessellated()
createSection()
createEntities()
createCircle()
createLine()

Fig 28

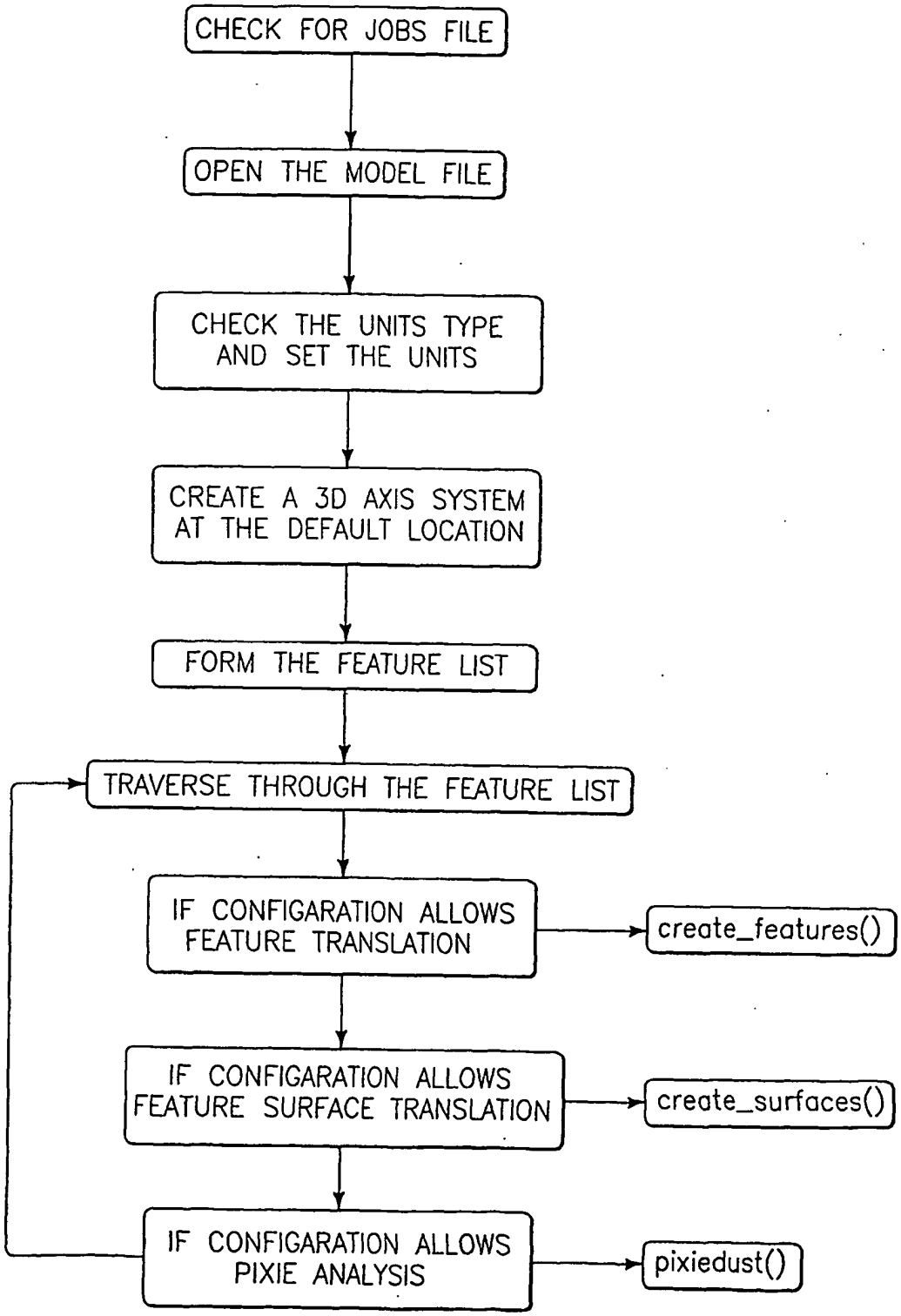

CHECK FOR JOBS FILE
OPEN THE MODEL FILE
CHECK THE UNITS TYPE AND SET THE UNITS
CREATE A 3D AXIS SYSTEM AT THE DEFAULT LOCATION
FORM THE FEATURE LIST
TRAVERSE THROUGH THE FEATURE LIST
IF CONFIGARATION ALLOWS FEATURE TRANSLATION
create_features()
IF CONFIGARATION ALLOWS FEATURE SURFACE TRANSLATION
create_surfaces()
IF CONFIGARATION ALLOWS PIXIE ANALYSIS
pixiedust()

Fig 29

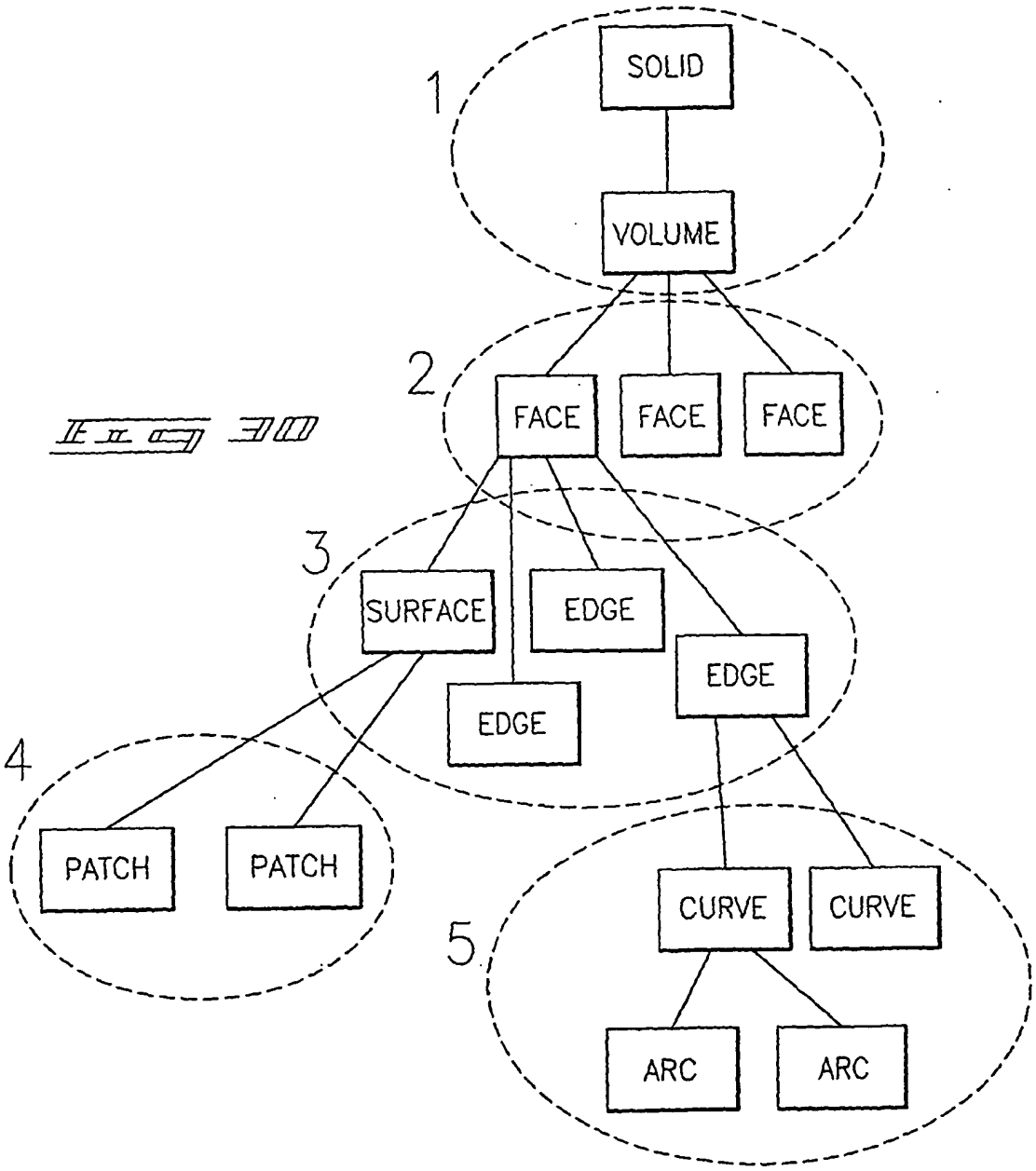

1
SOLID
VOLUME
2
FACE
FACE
FACE
3
SURFACE
EDGE
EDGE
EDGE
4
PATCH
PATCH
5
CURVE
CURVE
ARC
ARC

Fig 30

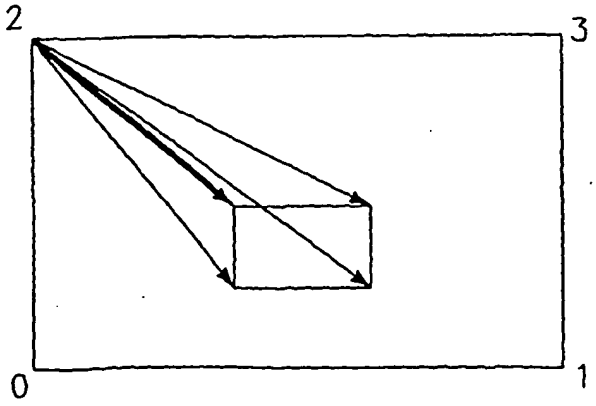

2
3
0
1

Fig 31

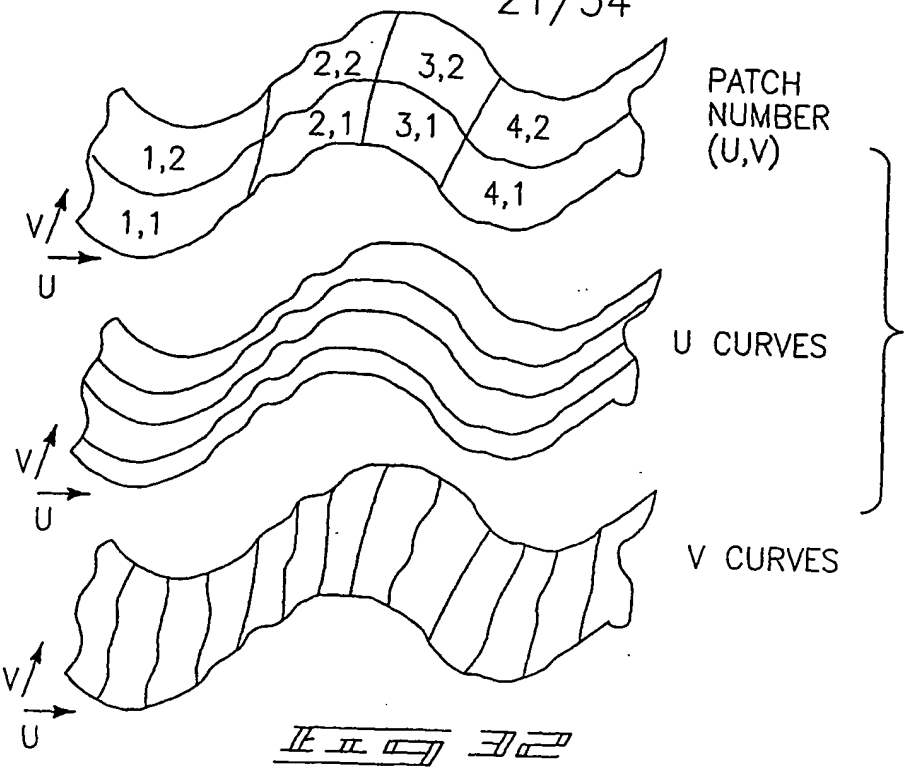


Fig 32

1. POINTS ARE IMPORTED FOR ONE CURVE AT A TIME.

2. CURVES ARE FIT THROUGH EACH SET OF POINTS.

3. SPLINES ARE THEN FIT THROUGH EACH OF THE CURVES.

4. THE RESULT IS A BLENDED SURFACE.

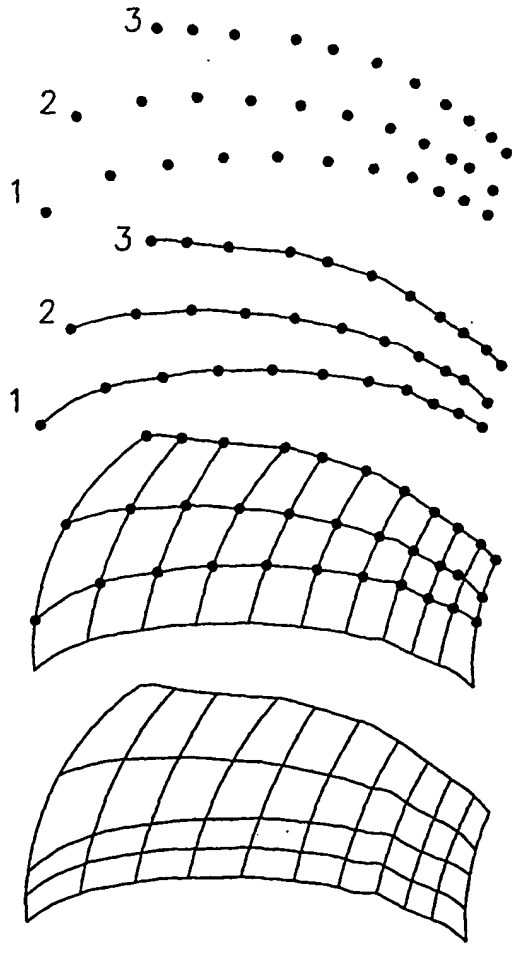


Fig 33

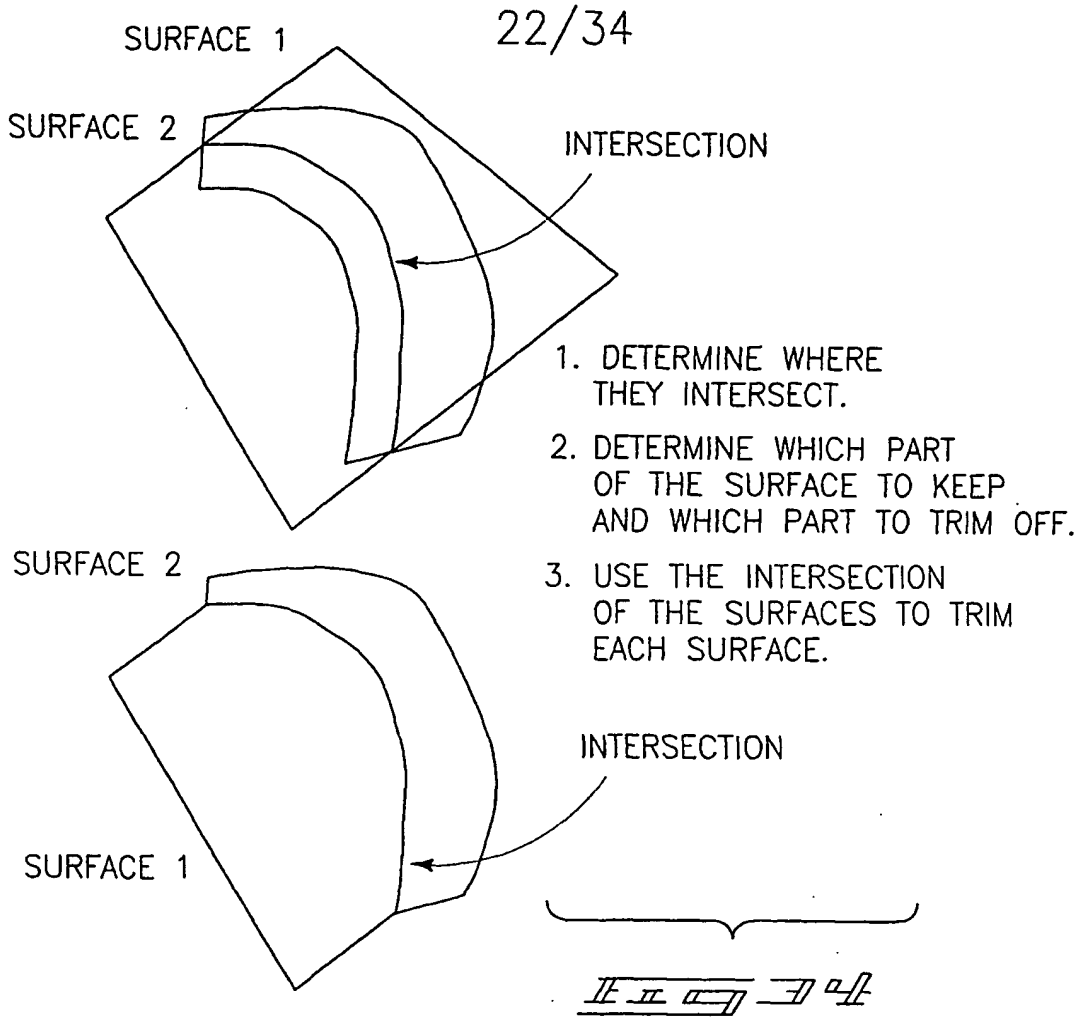

SURFACE 1
SURFACE 2
INTERSECTION
1. DETERMINE WHERE THEY INTERSECT.
2. DETERMINE WHICH PART OF THE SURFACE TO KEEP AND WHICH PART TO TRIM OFF.
3. USE THE INTERSECTION OF THE SURFACES TO TRIM EACH SURFACE.
SURFACE 2
INTERSECTION
SURFACE 1

Fig 34

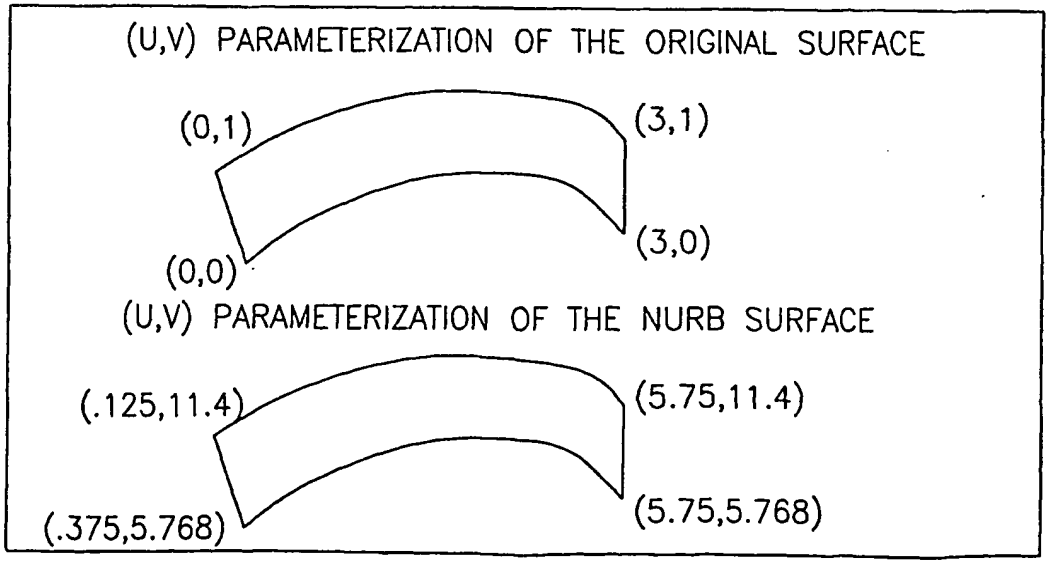

(U,V) PARAMETERIZATION OF THE ORIGINAL SURFACE
(0,1)
(3,1)
(3,0)
(0,0)
(U,V) PARAMETERIZATION OF THE NURB SURFACE
(.125,11.4)
(5.75,11.4)
(.375,5.768)
(5.75,5.768)

Fig 35

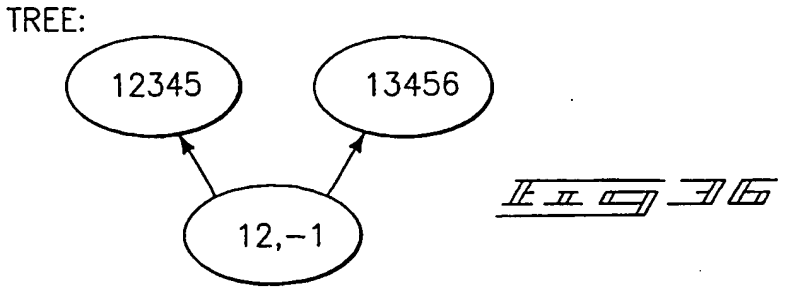


Fig 36

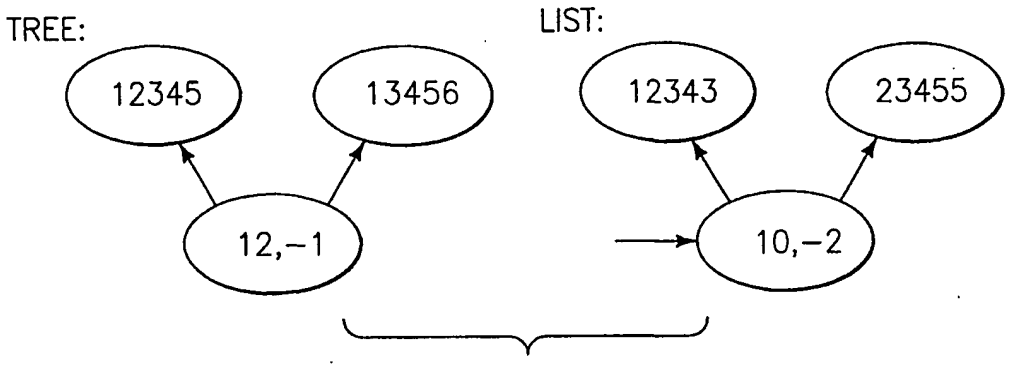


Fig 37

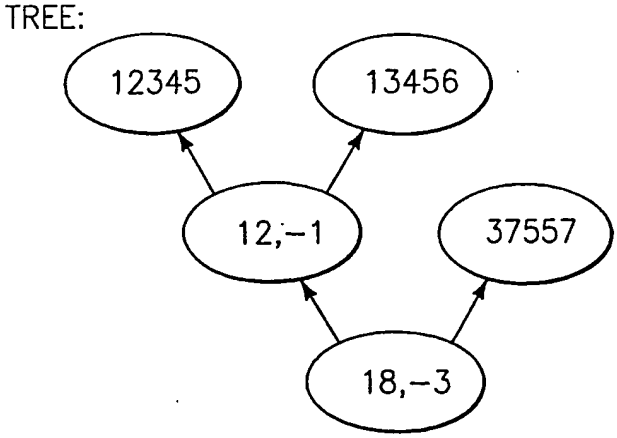


Fig 38

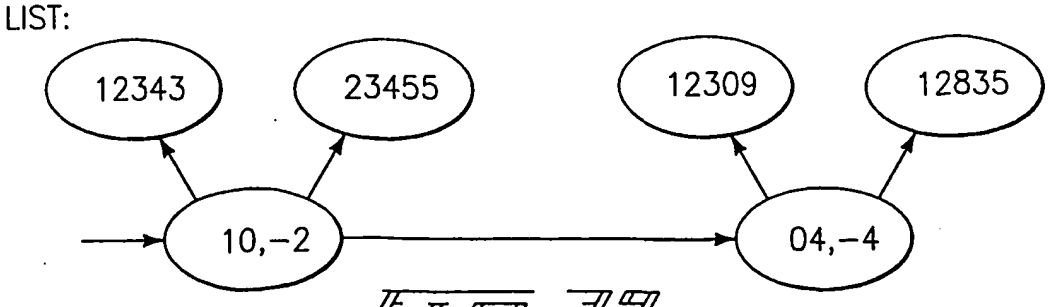


Fig 39

TREE:

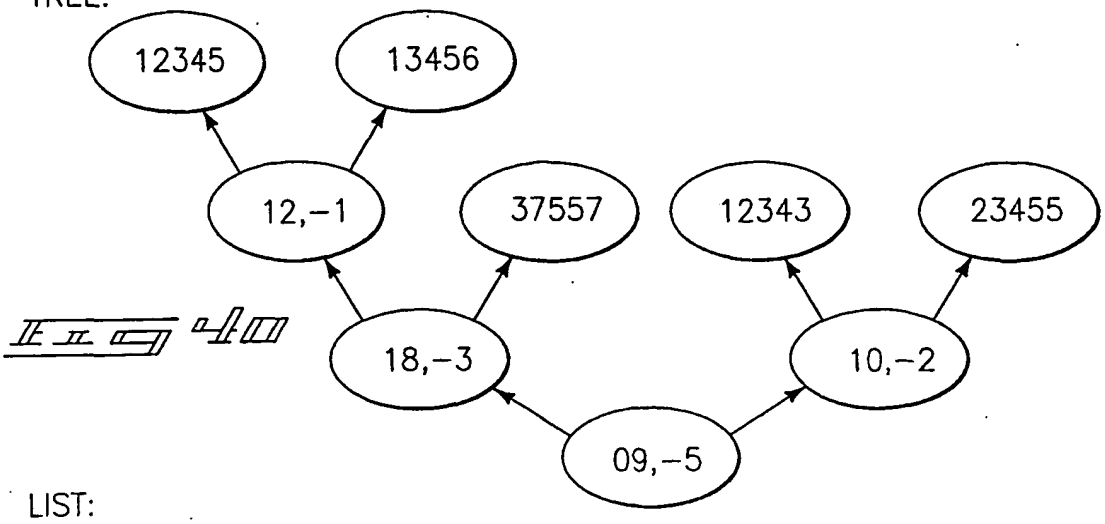


Fig 40

LIST:

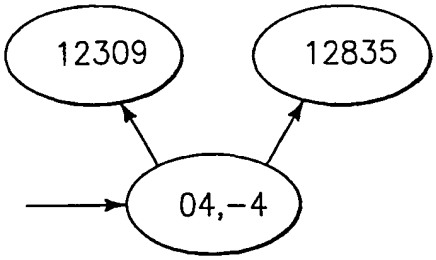


Fig 41

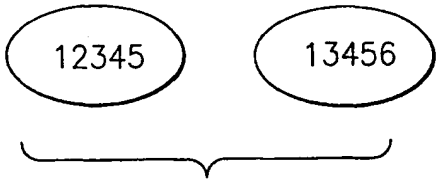


Fig 42

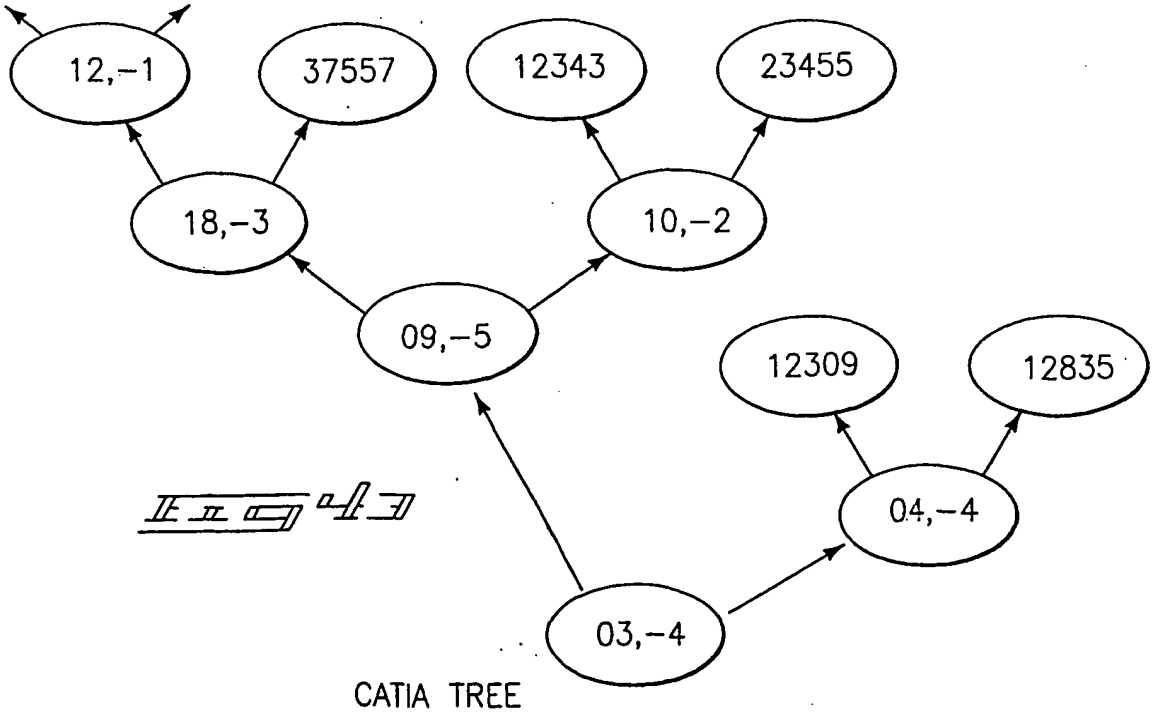


Fig 43

CATIA TREE

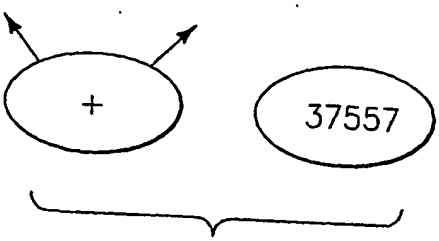
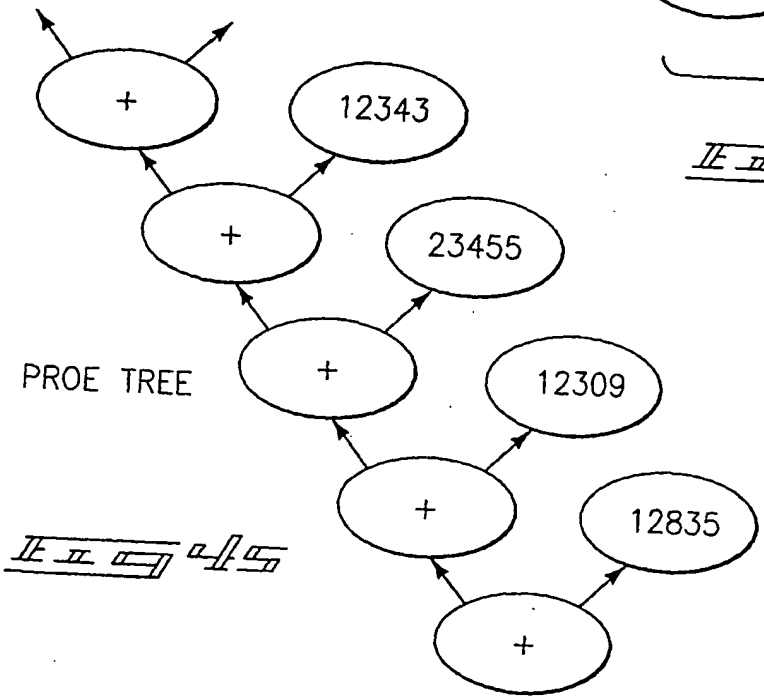


Fig 44



PROE TREE

Fig 45

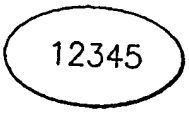


Fig 46

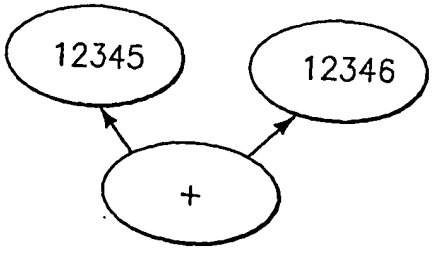
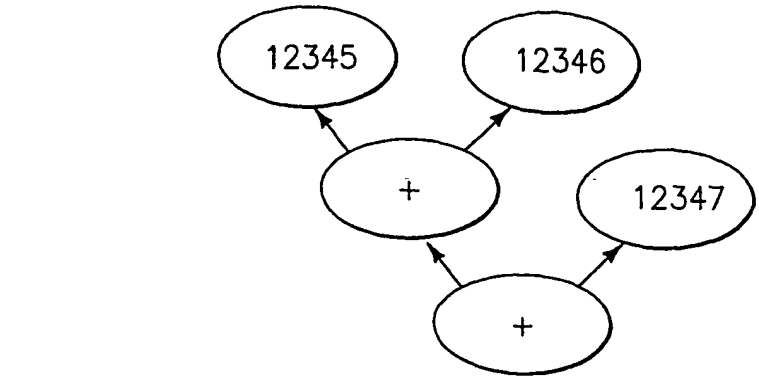


Fig 47


12345
12346
+
12347
+

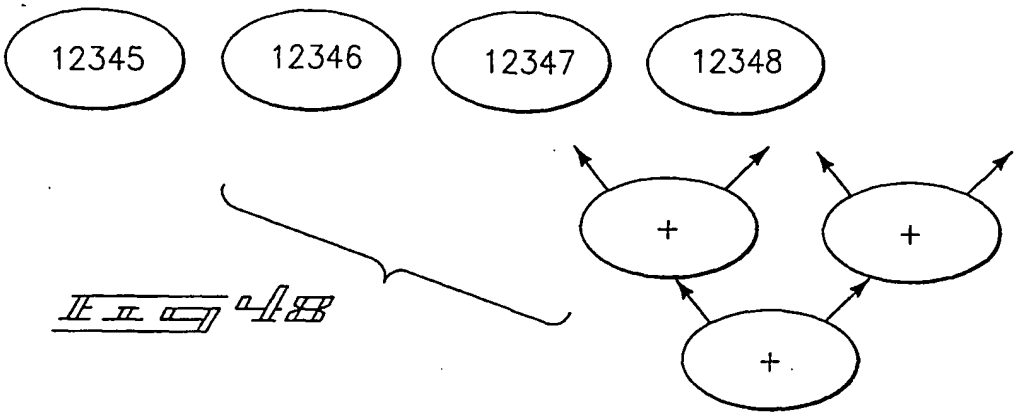

12345
12346
12347
12348
+
+
+

Fig 48

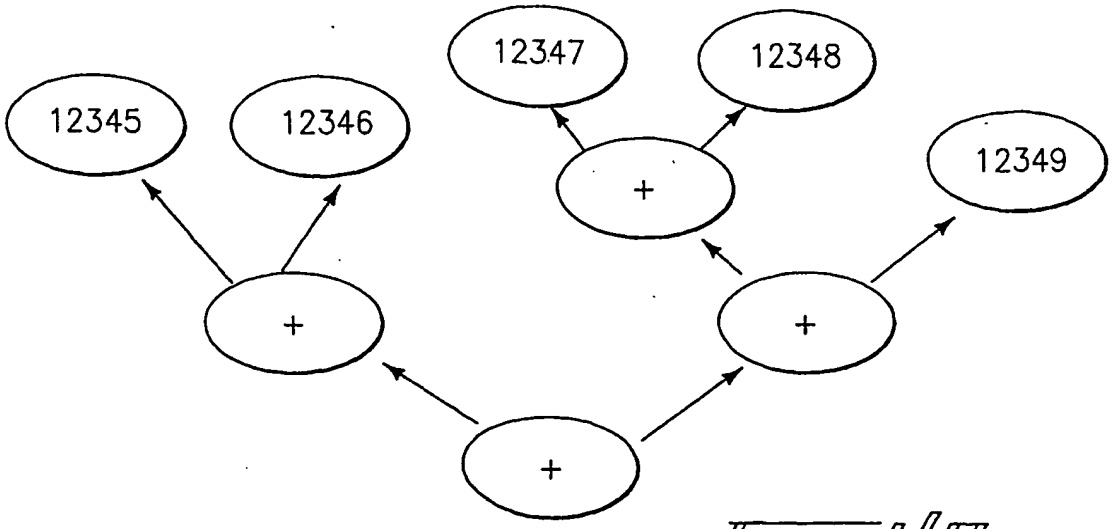

12345
12346
12347
12348
12349
+
+
+

Fig 49

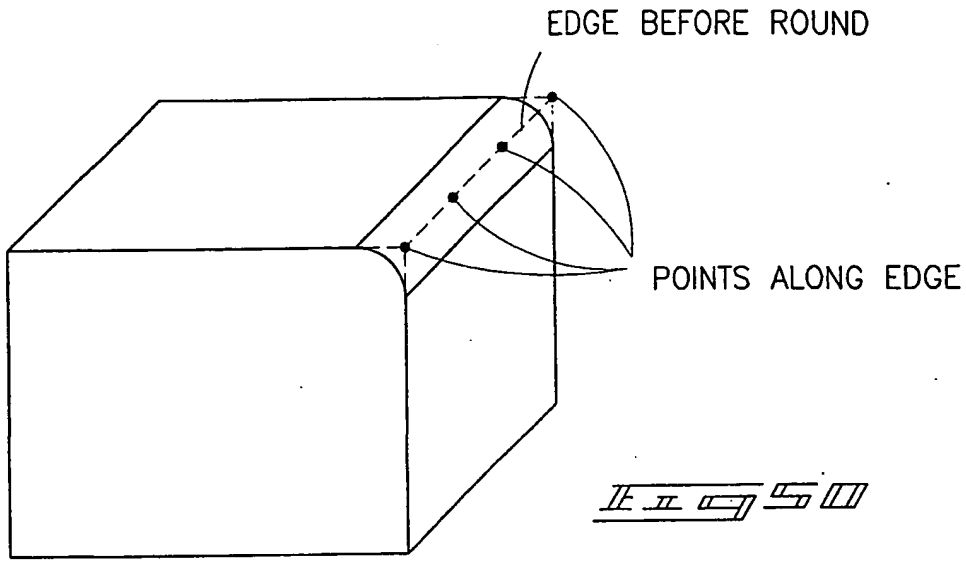


Fig 50

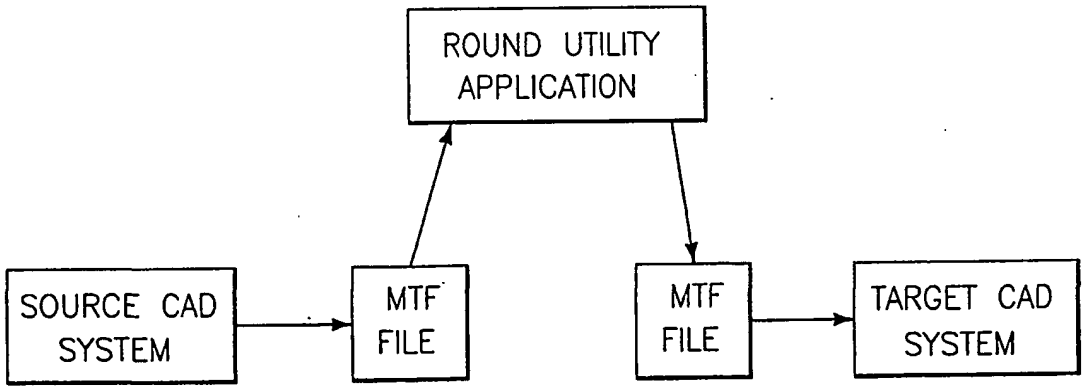


Fig 51

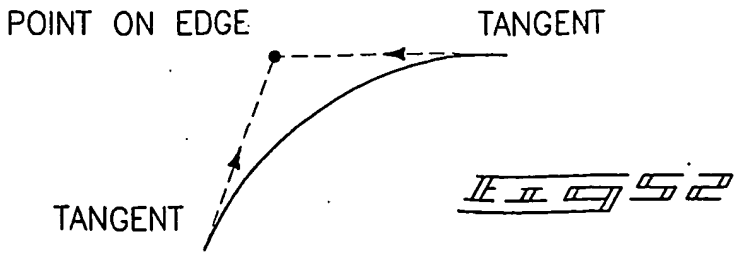
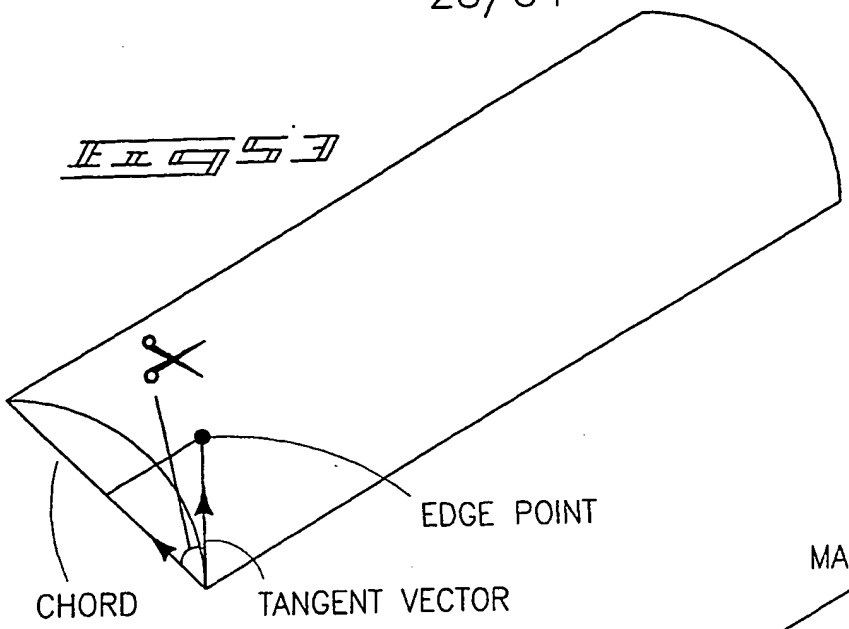


Fig 52

Fig 53



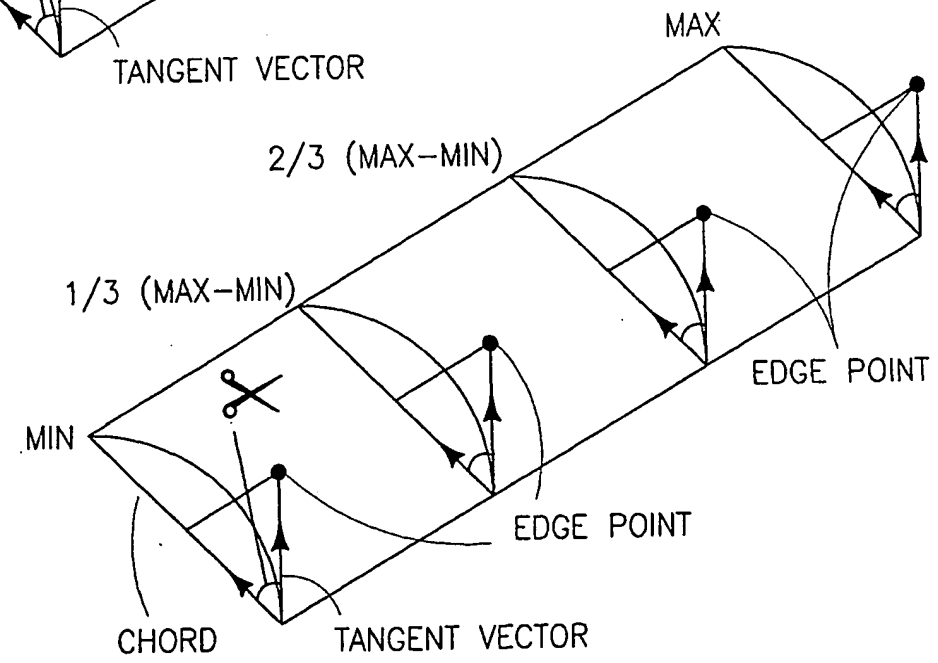


Fig 54

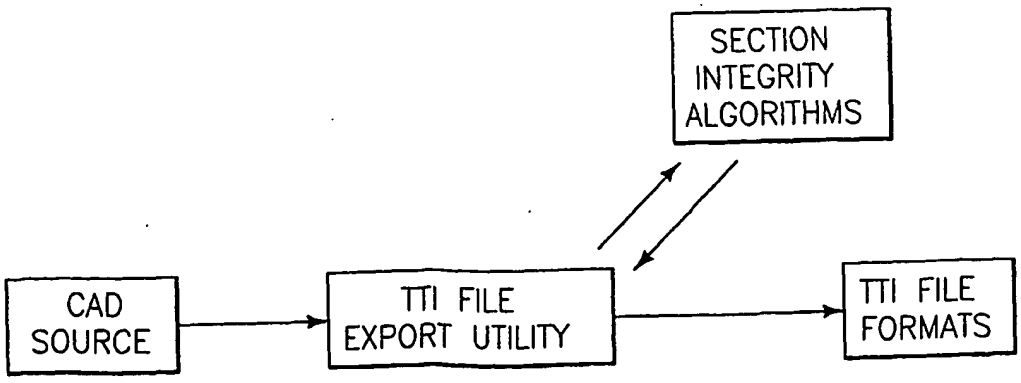


Fig 55

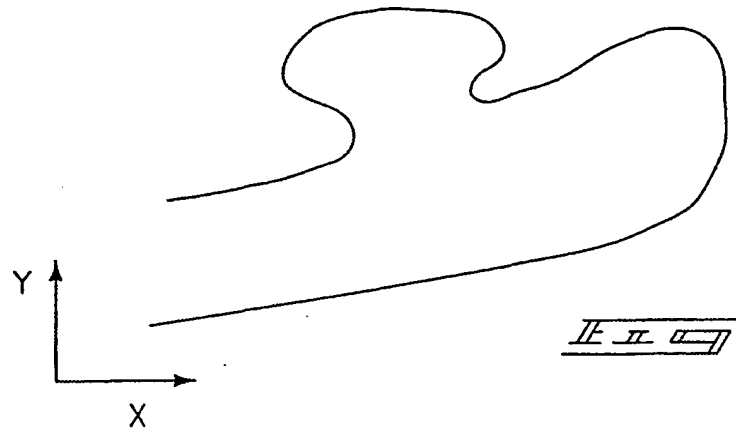


Fig 56

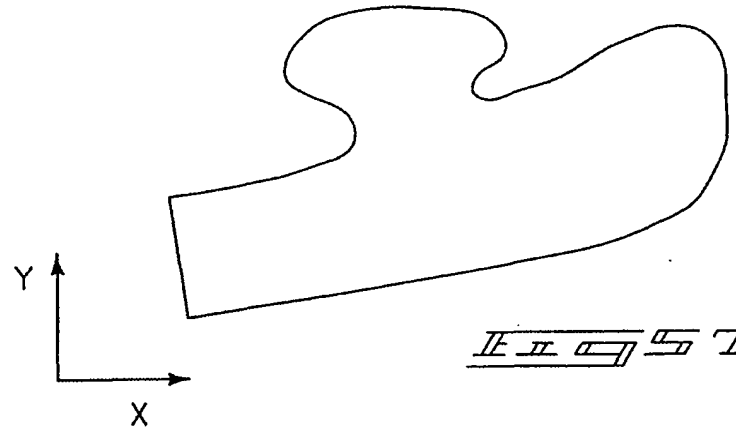


Fig 57

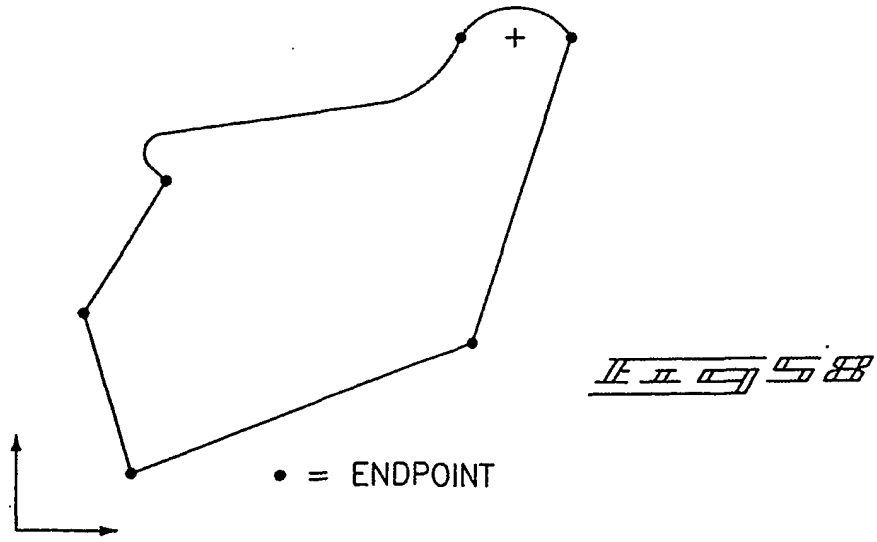


Fig 58

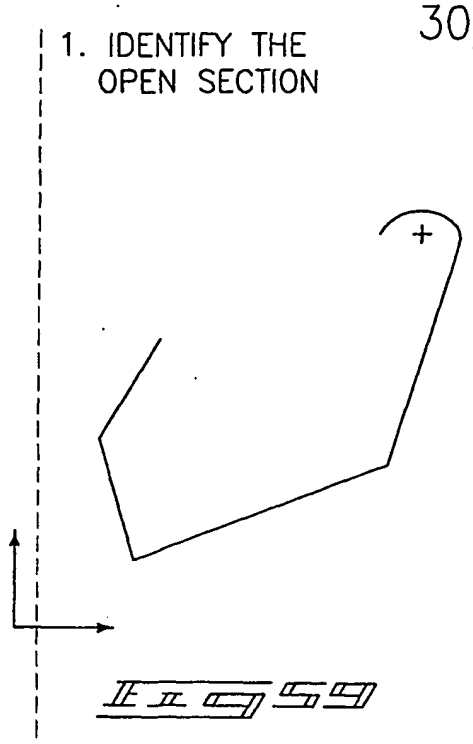


Fig 59

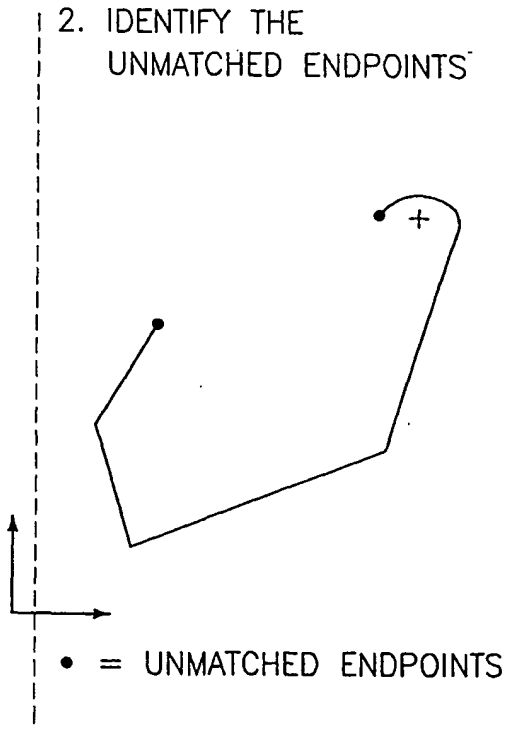


Fig 60

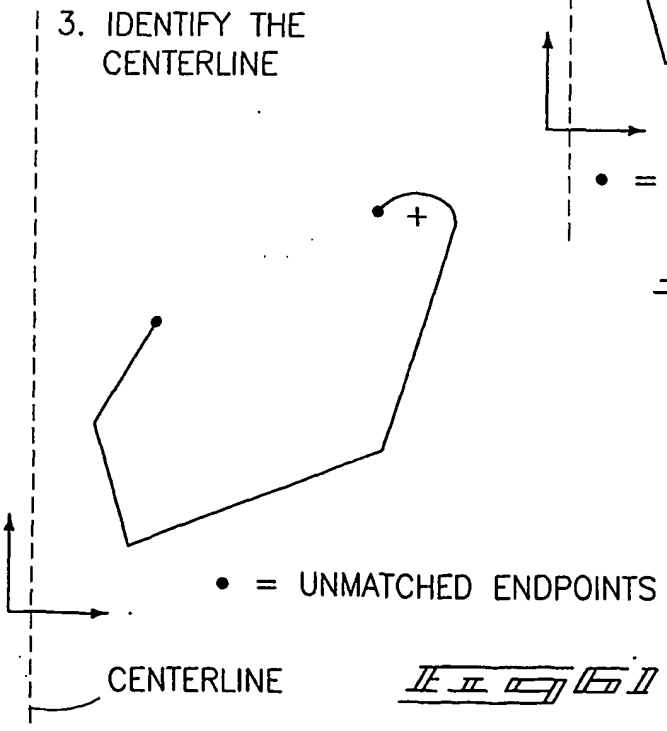


Fig 61

4. CONSTRUCT LINES THROUGH THE UNMATCHED ENDPOINTS AND PERPENDICULAR TO THE CENTERLINE

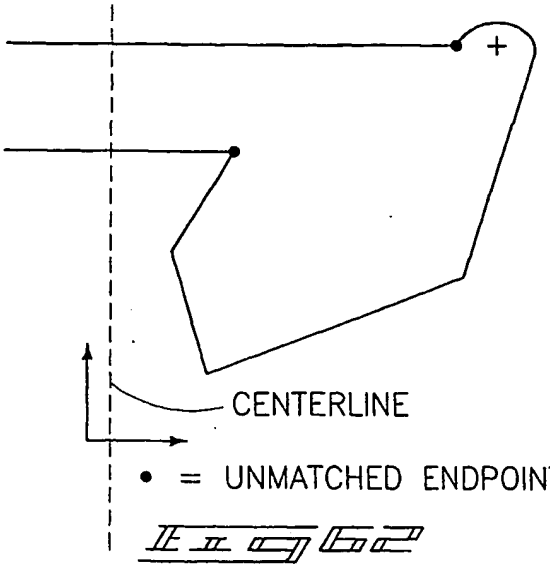


Fig 62

5. DETERMINE WHERE THESE NEW LINE SEGMENTS CROSS THE CENTERLINE

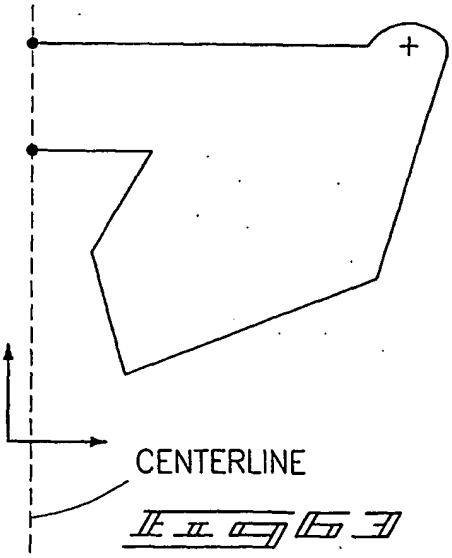


Fig 63

6. CONSTRUCT A LINE SEGMENT THROUGH THE INTERSECTION POINTS ALONG THE CENTERLINE TO COMPLETELY CLOSE THE SECTION

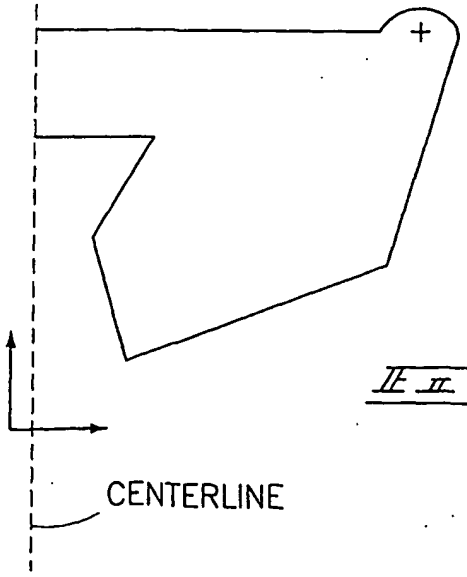


Fig 64

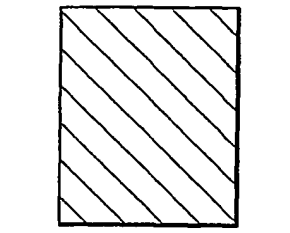


Fig 65

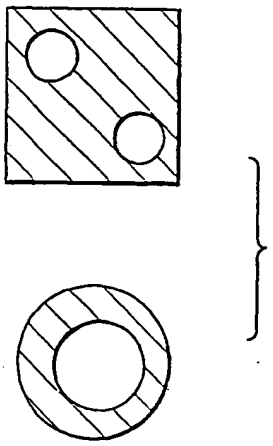


Fig 66

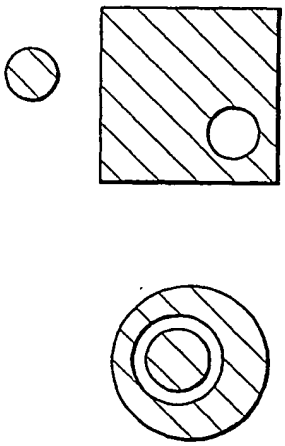


Fig 67

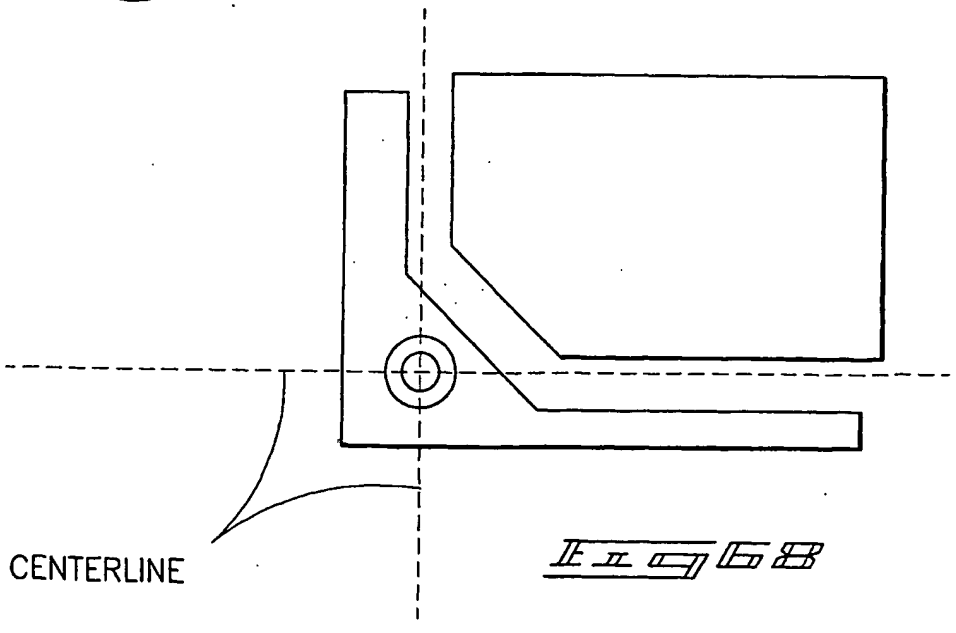


Fig 68

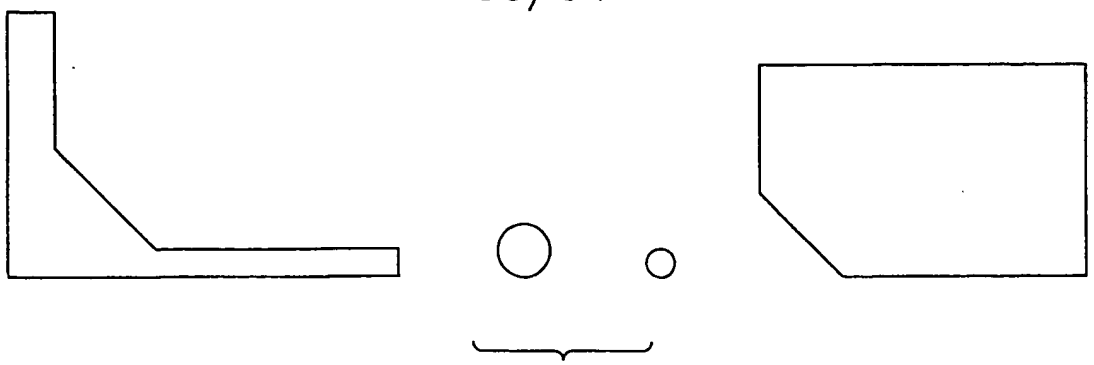


Fig 69

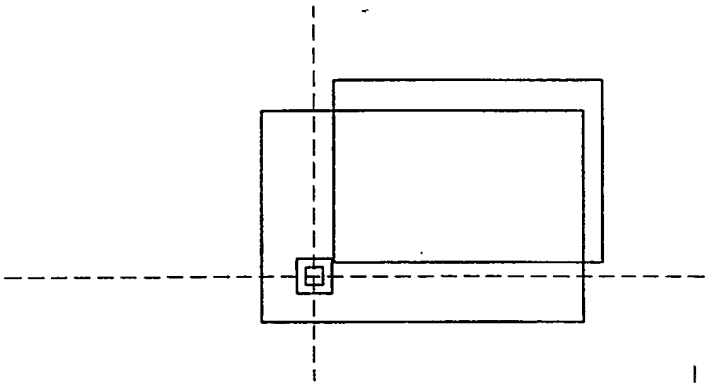


Fig 70

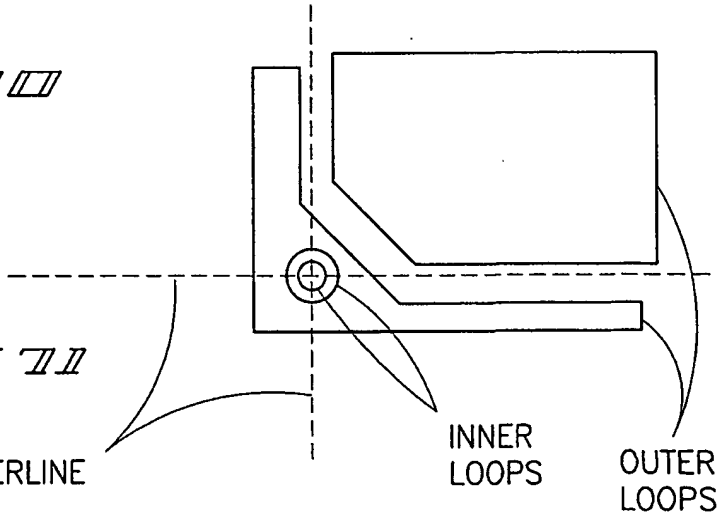


Fig 71

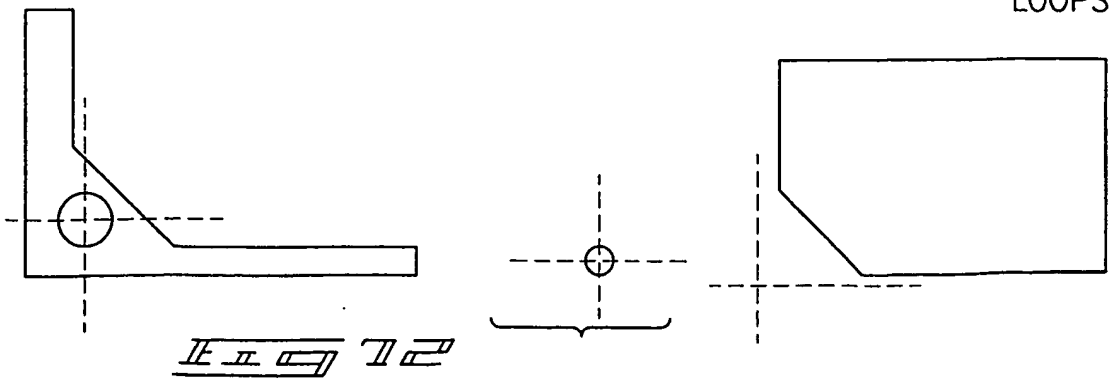


Fig 72

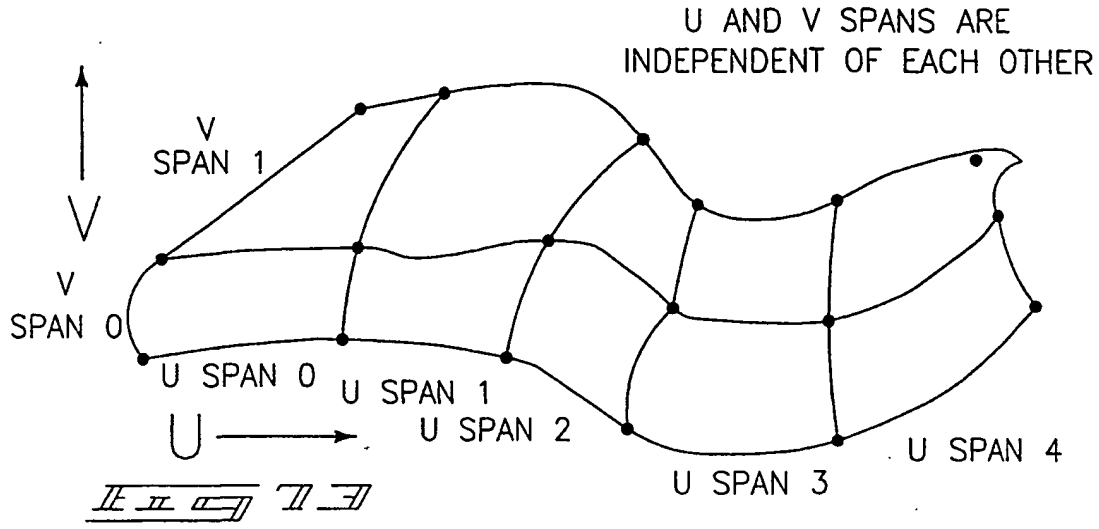


Fig 13

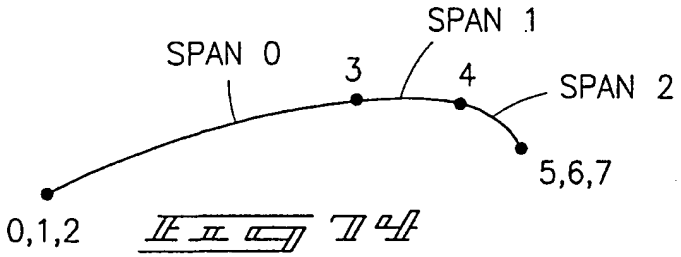


Fig 14

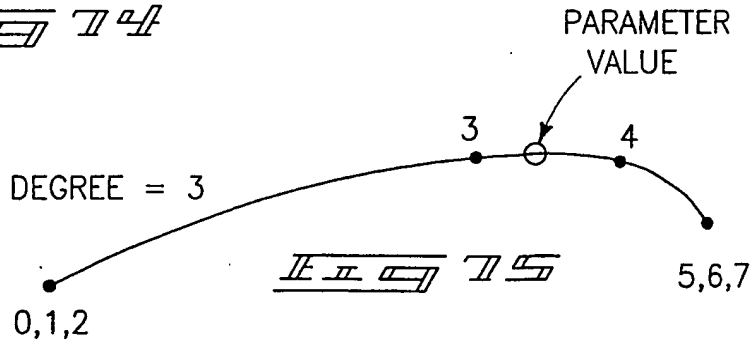


Fig 15

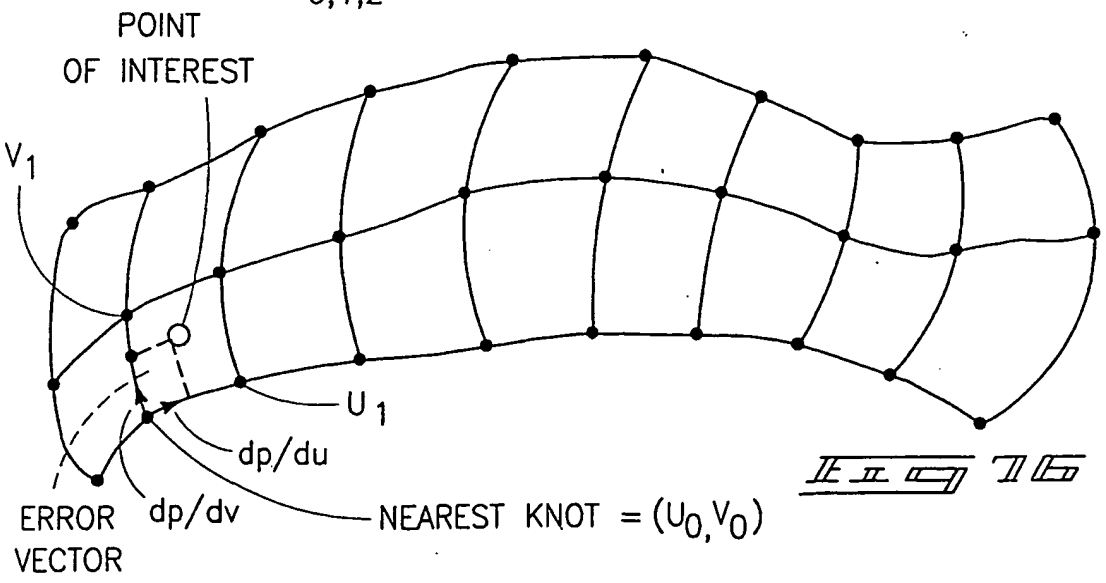


Fig 16

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/45501

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06K 9/54
US CL :382/306

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 382/306; 700/182; 706/919

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WEST
search terms: geometric model, discrepancy, mismatch, error

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4,831,549 A (RED et al) 16 May 1989, col. 4, lines 28-50.	1-64
A	US 5,615,317 A (FREITAG) 25 March 1997, Abstract.	1-64
A	US 5,691,909 A (FREY et al) 25 November 1997, Abstract.	1-64

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 20 FEBRUARY 2002	Date of mailing of the international search report 21 MAR 2002
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer Stephen Brinich Telephone No. (703) 305-4390

Form PCT/ISA/210 (second sheet) (July 1998)*